Studies
on
Local Search Approaches
to One Dimensional Cutting Stock Problems

Shunji Umetani

STUDIES
ON
LOCAL SEARCH APPROACHES
TO ONE DIMENSIONAL CUTTING STOCK PROBLEMS

Shunji UMETANI

Submitted in partial fulfilment of
the requirement for the degree of
DOCTOR OF INFORMATICS
(Applied Mathematics and Physics)

KYOTO UNIVERSITY
KYOTO, JAPAN
JANUARY, 2003

Preface

As combinatorial optimization problems appear in important application areas, a wide variety of algorithms has been developed for several decades. Unfortunately, most of these combinatorial optimization problems are known to be NP-hard, i.e., they belong to a class of intractable problems. Therefore, under the widely believed conjecture P $\neq$ NP, their exact algorithms must be exhaustively time consuming. However, in most real applications, we are satisfied with good solutions obtained in reasonable computational time, even if we are not able to obtain an exact optimal solution. In this sense, to deal with large instances of such intractable combinatorial optimization problems, *approximate* (or *heuristic*) *algorithms* are very important, and have been intensively studied in recent years.

The *local search* and its extensions called *metaheuristics* are part of the most representative approaches for this purpose. These algorithms have been applied to many intractable problems for their simplicity and flexibility, and succeeded in obtaining good solutions to some extent. However, in recent years, due to the diversification and sophistication of systems in real applications, these problems have become more intractable so that simple applications of metaheuristic algorithms are not sufficient. To overcome these difficulties, we need to construct meaningful mathematical models based on careful analysis of the problems, and develop more powerful and/or flexible tools.

In the thesis, we consider one of such intractable combinatorial optimization problems, called the *one dimensional cutting stock problem* (1D-CSP), which have been intensively studied since 1960s because of its wide applicability to material industries. In their earlier studies, where minimizing the cost associated with materials was one of most important issues, 1D-CSP have been studied as a variant of integer linear programming problems (ILP) and many useful algorithms were developed. However, in recent industry, as the cost associated with setup costs have become more dominant than the cost associated with materials, the solutions

of such algorithms become not desirable to users. We propose other formulations of 1D-CSP in which we include the number of setups as an input parameter, and design local search algorithms based on various heuristics and/or mathematical programming techniques. Our approach gives users useful information of the trade-off curves between these cost functions by controlling the input parameter.

The studies on 1D-CSP are still developing, and many important problems remain to be solved in this field. The author hopes that the research gives useful tools and ideas to conquer these intractable combinatorial optimization problems.

January, 2003

Shunji Umetani

Acknowledgment

This thesis would not have been possible without the help of many others. First of all, I am heartily grateful to Professor Toshihide Ibaraki of Kyoto University for his enthusiastic guidance, discussion and persistent encouragement. He commented in detail on the whole work in the manuscript, which significantly improved the accuracy of the arguments and quality of the exposition. Without his considerable help, none of this work could have been completed.

A great deal of gratitude goes to Professor Mutsunori Yagiura of Kyoto University with whom I wrote joint papers. His great collaboration was very important in forming the idea of this thesis.

I also wish to express my gratitude to Professor Koji Nonobe of Kyoto University, Professor Toshimasa Ishii of Toyohashi University of Technology, Professor Takashi Horiyama of Kyoto University, Professor Hirotaka Ono of Kyushu University, Professor Liang Zhao Utsunomiya University, and all members in Professor Ibaraki's laboratory for many enlightening discussions on the area of this work. I especially mention a member who gave me useful comments, namely Shinji Imahori.

My deepest gratitude is to my family for their heartfelt cooperation and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Combinatorial Optimization Problems

An optimization problem is generally defined as follows:

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & \boldsymbol{x} \in F, \end{aligned} \tag{1.1}$$

where F is the set of solutions $\boldsymbol{x}$ which satisfy all the given constraints. F is called the *feasible region* and each $\boldsymbol{x} \in F$ is called a *feasible solution*. The function f is called the *objective function* (or *cost function*), and a feasible solution $\boldsymbol{x}^* \in F$ is *optimal* if $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ holds for all other feasible solutions $\boldsymbol{x} \in F$. We call an optimization problem (1.1) a *combinatorial optimization problem* if F is combinatorial in some sense. We often encounter combinatorial optimization problems in many real applications, e.g., production planning [20], machine scheduling [8][4], plant location [5], vehicle routing [33][61], crew scheduling [53][10][25], frequency assignment [82][66], VLSI design [2], etc. Other recent application areas include problems in molecular biology, architecture and their importance has been widely recognized. Many of these real problems can be described as extensions of the following combinatorial optimization problems.

Bin Packing Problem (BPP)

Input: A set of items $U = \{u_1, u_2, \ldots, u_m\}$, their sizes $s(u_i)$ for each $u_i \in U$, and the bin capacity B.

Output: A partition of U into the minimum number n of disjoint sets $U_1, U_2, \ldots, U_n$ such that the total size $\sum_{i \in U_j} s(u_i)$ of items in each U_j is B or less.

Graph Coloring Problem (GCP)

Input: An undirected graph $G = (V, E)$.

Output: A coloring $\pi : V \to \{1, \ldots, \chi\}$ of all vertices $v_i \in V$ with the minimum number of colors χ, in which the end of vertices v_i, v_j of each edge $\{v_i, v_j\} \in E$ has different colors $\pi(v_i) \neq \pi(v_j)$.

Knapsack Problem (KP)

Input: A set of items $U = \{u_1, u_2, \ldots, u_m\}$, their sizes $s(u_i)$ and values $v(u_i)$ for each $u_i \in U$, and the knapsack capacity B.

Output: A subset $U' \subseteq U$ of the maximum total value $\sum_{u_i \in U'} v(u_i)$ of items in U' such that the total size $\sum_{i \in U'} s(u_i)$ of items is B or less.

Set Covering Problem (SCP)

Input: A finite set S and a collection $C = \{S_1, S_2, \ldots, S_m\}$ of subsets $S_i \subseteq S$.

Output: A set cover for S, i.e., a collection $C' \subseteq C$ of the minimum cardinality $|C'|$ such that every element in S belongs to at least one member $S_i \in C'$.

Traveling Salesman Problem (TSP)

Input: A set of cities $C = \{c_1, c_2, \ldots, c_m\}$ and distances $d(c_i, c_j)$ for each pair of cities $c_i, c_j \in C$.

Output: A tour $\sigma : \{1, \ldots, m\} \to C$ of all cities $c_i \in C$ of the minimum total length $\sum_{i=1}^{m} d(\sigma(i), \sigma(i+1)) + d(\sigma(m), \sigma(1))$.

For example, vehicle routing problem (VRP) is described as follow: there are a number of customers to be served from a unique depot. Each customer has a quantity of goods and a number of time periods (time windows) to be delivered. Each vehicle has the capacity on the goods carried. VRP asks to design routes of vehicles with the minimum total cost such that all goods are delivered to customers in their time windows. The main costs of VRP are the number of vehicles used, and the total distance traveled. In this sense, VRP contains BPP and a number of TSPs with time windows, i.e., the problem of determining the number of

vehicles is formulated as BPP, and the problem of determining the route of each vehicle is formulated as a special TSP with time windows.

Another example is airline crew scheduling problem (ACSP): Given a schedule of flights for a particular aircraft type. Each crew is assigned for a set of flights (a weekly schedule) satisfying many constraints such as limited flying time, minimum rests between flights, returning to starting point after a set of flights, etc. ACSP asks to assign crews for weekly schedules such that all flights are covered by at least a given number of crews. The objective is to minimize the amount paid to the crews. ACSP can be formulated as SCP, where the set of all flights correspond to the set S and all possible weekly schedules correspond to all subsets $S_i \in C$.

Most of these combinatorial optimization problems (e.g. the above all problems) are known to be *NP-hard*, and it is unlikely that there exist polynomial time algorithms for NP-hard problems [31]. One of the representative approaches is that it formulates these combinatorial optimization problems as *integer linear programming problem* (ILP) and solves them by efficient exact algorithms, e.g., *branch-and-bound* or *dynamic programming* algorithm [72][99].

$$\begin{aligned} \text{(ILP)} \quad \text{minimize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \text{ for } i = 1, \ldots, m \\ & x_j \in \boldsymbol{Z}_+, \text{ for } j = 1, \ldots, n. \end{aligned} \tag{1.2}$$

The *mixed-integer programming problem* (MILP) is a superset of ILP, in that some of the variables are allowed to be nonnegative continuous values. Although a variety of efficient branch-and-bound solvers have been developed as general purpose tools, many combinatorial optimization problems still remain to be intractable because of the following reasons:

- A solution is required rapidly, within a few second or minutes.
- The instance is so large and/or complicated that it can not be formulated as ILP or MILP of reasonable size.
- Even though it has been formulated as an ILP (or MILP), it is difficult or impossible for these exact algorithms to find optimal solutions.

- For certain combinatorial optimization problems such as TSP, it is easy to find feasible solutions by inspection or knowledge of the problem structure, and a general purpose integer programming approach is ineffective.

Fortunately, in most applications, we are satisfied with good solutions obtained in reasonable computational time even if we are not able to obtain an exact optimal solution. In this sense, to deal with large instances of such intractable combinatorial optimization problems, *approximate* or *heuristic* algorithms are important and such approaches have been intensively studied in recent years.

1.2 Local Search and Metaheuristics

The basic ideas of approximate algorithms are the *greedy algorithm* and the *local search algorithm* (LS) [74][1]. The greedy algorithm is a constructive method that constructs an approximate solution step by step on the basis of the local information. For example, the nearest neighbor method [64] is a greedy algorithm for TSP, which starts from an arbitrary city, and repeatedly moves to the nearest unvisited city until all cities are visited. On the other hand, the local search algorithm is an improvement method that iteratively modifies the current solution to obtain a better solution until no better solution is obtained by its modification. For example, the 2-exchange (2-opt) heuristic [74][64][57] is a local search algorithm for TSP, which repeatedly exchanges a pair of crossing edges of the tour until no better tour is obtained by the operation.

The local search starts from an initial feasible solution and repeatedly replaces it with a better solution in its *neighborhood* $N(\boldsymbol{x})$ until no better solution is found in $N(\boldsymbol{x})$, where $N(\boldsymbol{x})$ is a set of solutions obtainable from $\boldsymbol{x}$ by applying a slight perturbation. A solution $\boldsymbol{x}$ is called *locally optimal*, if no better solution $\boldsymbol{x}'$ is found in $N(\boldsymbol{x})$. The simple local search algorithm is described as follow:

Algorithm Local Search (LS)

Input: The initial feasible solution $\boldsymbol{x}^{init}$.

Output: A locally optimal solution $\boldsymbol{x}$.

Step 1: Set $\boldsymbol{x} := \boldsymbol{x}^{init}$.

Step 2: If there is a feasible solution $\boldsymbol{x}' \in N(\boldsymbol{x})$ such that $f(\boldsymbol{x}') < f(\boldsymbol{x})$ holds, set $\boldsymbol{x} := \boldsymbol{x}'$ and return to Step 1. Otherwise (i.e., $f(\boldsymbol{x}') \geq f(\boldsymbol{x})$ holds for all $\boldsymbol{x}' \in N(\boldsymbol{x})$) output $\boldsymbol{x}$ and halt.

The search procedure of finding the next solution $\boldsymbol{x}' \in N(\boldsymbol{x})$ is called the *neighborhood search*, and the set of all solutions which may be potentially visited in a local search algorithm is called the *search space*.

To design a local search algorithm for a combinatorial optimization problem, we must consider the following ingredients:

(i) How to generate an initial feasible solution $\boldsymbol{x}^{init}$.

(ii) How to construct the neighborhood $N(\boldsymbol{x})$ from the current solution $\boldsymbol{x}$.

(iii) How to compute the objective function $\boldsymbol{x}$ efficiently.

(iv) How to specify the order of solutions to be evaluated in the neighborhood $N(\boldsymbol{x})$, and the selection rules of the next solution.

Although the local search algorithm is powerful for its simplicity, its naive implementation does not attain sufficiently good solutions since it only visits a small number of new solutions. To overcome this, many variants of the local search algorithm have been developed, where their strategies are:

Initial solution: executing a number of the local search algorithms from different initial solutions, e.g., multi-start local search (MLS) [60][65], iterated local search (ILS) [56][69], greedy randomized adaptive search procedure (GRASP) [27][63], variable neighborhood search (VNS) [71][51], genetic algorithm (GA) [54][42], scatter search [37][38], and ant colony system [19][30].

Neighborhood: adopting to a larger neighborhood or a sophisticated structure of neighborhood, e.g., variable depth search (VDS) [60][65], very large-scale neighborhood search[3], ejection chain [39][75][9][100].

Move strategy: allowing to move worse solutions and controlling moves by a randomized or sophisticated strategy, e.g., simulated annealing (SA) [62][55], threshold accepting (TA) [21], great deluge algorithm (GDA) [22], tabu search (TS) [36][41].

Search space: adopting a search space different from F (i.e., including infeasible region) and modifying the objective function f so that we can evaluate the amount of infeasibility of solutions, e.g., strategic oscillation [40][50]. (This approach often appears in many local search algorithms since finding a feasible solution is not easy for many combinatorial optimization problems.)

Objective function: adaptively perturbating the objective function in order to escape poor locally optimal solutions, e.g., guided local search [86][96], noising method [13][16], search space smoothing method [45], Lagrangian-based heuristics [28][6][11][10].

The framework of these variants of the local search algorithms are called *metaheuristics*. Some representative algorithms in metaheuristics stated as follows:

The iterated local search (ILS) repeats the local search from different initial solutions, where they are generated by random perturbations applied to the best solution obtained by then.

Algorithm Iterated Local Search (ILS)

Input: The first initial solution $\boldsymbol{x}^{init}$.

Output: The best solution $\boldsymbol{x}^*$.

Step 1: Set $\boldsymbol{x}^* := \boldsymbol{x}^{init}$.

Step 2: Generate an initial solution $\boldsymbol{x}$ of the next local search algorithm by slightly perturbating $\boldsymbol{x}^*$ randomly.

Step 3: Apply the local search algorithm (LS) to obtain a locally optimal solution $\boldsymbol{x}$.

Step 4: If $f(\boldsymbol{x}) < f(\boldsymbol{x}^*)$ holds, set $\boldsymbol{x}^* := \boldsymbol{x}$. If some stopping criteria are satisfied, output $\boldsymbol{x}^*$ and halt; otherwise return to Step 2.

The genetic algorithm (GA) employs evolutionary process in nature. GA repeatedly generates the set of new candidate solutions $N(P)$ by applying the operations such as *crossover*, *mutation* and *selection* to the set of candidate solution P. A crossover generates one or more solutions by combining two or more candidate solutions, and a mutation generates a solution by slightly perturbating a candidate solution. GA starts from an initial set of candidate solutions P and repeatedly replaces P with $P' \subseteq P \cup N(P)$ according to its selection rules.

Algorithm Genetic Algorithm (GA)

Input: The set of initial candidate solutions P.

Output: The best solution $\boldsymbol{x}^*$.

Step 1: Construct an initial set of candidate solutions P. Let $\boldsymbol{x}^*$ be the best solution among P.

Step 2: Repeat the following Step 2-1 and/or Step2-2 to obtain the set of new candidate solutions $N(P)$ from the current set of solutions P.

Step 2-1: Crossover two or more candidate solutions to generate a new candidate solution.

Step 2-2: Mutate a candidate solution to generate a new candidate solution.

Step 3: If a solution $\boldsymbol{x}$ with $f(\boldsymbol{x}) < f(\boldsymbol{x}^*)$ is found in Step 2, set $\boldsymbol{x}^* := \boldsymbol{x}$.

Step 4: Select the set of a given number of candidate solutions P' from the resulting $P \cup N(P)$, and set $P := P'$.

Step 5: If some stopping criteria are satisfied, output $\boldsymbol{x}^*$ and halt; otherwise return to Step 2.

The simulated annealing (SA) is a kind of probabilistic local search, in which test solutions are randomly chosen from $N(\boldsymbol{x})$ and accepted with probability that is 1 if the test solution is better than the current solution $\boldsymbol{x}$, and positive even if the test solution is worse than the current solution $\boldsymbol{x}$. By allowing moves to worse solutions, SA can escape from poor locally optimal solutions. The acceptance probability is controlled by a parameter called *temperature*, whose idea stems from the physical annealing process.

Algorithm Simulated Annealing (SA)

Input: The initial solution $\boldsymbol{x}^{init}$, and the initial temperature t.

Output: The best solution $\boldsymbol{x}^*$.

Step 1: Set $\boldsymbol{x}^* := \boldsymbol{x}^{init}$, and $\boldsymbol{x} := \boldsymbol{x}^{init}$.

Step 2: Generate a solution $\boldsymbol{x}' \in N(\boldsymbol{x})$ randomly, and set $\Delta := f(\boldsymbol{x}') - f(\boldsymbol{x})$. If $\Delta < 0$ holds (i.e., a better solution is found), set $\boldsymbol{x} := \boldsymbol{x}'$; otherwise set $\boldsymbol{x} := \boldsymbol{x}'$ with probability $e^{-\Delta/t}$.

Step 3: If $f(\boldsymbol{x}) < f(\boldsymbol{x}^*)$ holds, set $\boldsymbol{x}^* := \boldsymbol{x}$. If some stopping criteria are satisfied, output $\boldsymbol{x}^*$ and halt; otherwise update the temperature t according to some rules and return to Step 2.

The tabu search (TS) tries to enhance the local search by the memory of previous searches. TS repeatedly replaces the current solution $\boldsymbol{x}$ by its best neighbor $\boldsymbol{x}' \in N(\boldsymbol{x})$ even if $f(\boldsymbol{x}') \geq f(\boldsymbol{x})$ holds. To avoid cycling of solutions, TS restricts the neighborhood $N(\boldsymbol{x}) \setminus (\{\boldsymbol{x}\} \cup T)$ by a *tabu list* T which is design to prevent TS going back to recently visited solutions.

Algorithm Tabu Search (TS)

Input: The initial solution $\boldsymbol{x}^{init}$.

Output: The best solution $\boldsymbol{x}^*$.

Step 1: Set $\boldsymbol{x}^* := \boldsymbol{x}^{init}$, $\boldsymbol{x} := \boldsymbol{x}^{init}$, and $T := \emptyset$.

Step 2: Find the best solution $\boldsymbol{x}' \in N(\boldsymbol{x}) \setminus \{\boldsymbol{x} \cup T\}$, and set $\boldsymbol{x} := \boldsymbol{x}'$.

Step 3: If $f(\boldsymbol{x}) < f(\boldsymbol{x}^*)$ holds, set $\boldsymbol{x}^* := \boldsymbol{x}$. If some stopping criteria are satisfied, output $\boldsymbol{x}^*$ and halt; otherwise update T according to some rules and return to Step 2.

Many approaches in metaheuristics are based on the analogies with processes and disciplines in nature such as statistical physics, biological evaluation, etc. More details about local search and metaheuristics are found in [77][73][1][102].

Theoretically, neither nontrivial bounds on the quality of local optimal solutions nor nontrivial bounds on the time complexity of local search have been known. However, in practice, many local search algorithms are successful to obtain sufficiently good solutions in reasonable computational time. One of the attractive features of local search and metaheuristics consists in its simplicity and robustness. We can develop local search and metaheuristic algorithms without knowing detailed mathematical properties of the problem, and still attain reasonably good solutions in practically feasible time [101]. Another good feature causes from its flexibility, i.e., we can much improve their performance by introducing sophisticated data structures and effective heuristics of the problem.

1.3 One Dimensional Cutting Stock Problems

The *cutting* and *packing* problem models the practical problem of considering how to arrange the small items in the large items. In this sense, the following problems are essentially the same except for kind of assignment, assortment, dimensionality, shapes, etc.: *knapsack problem, bin packing problem, cutting stock problem, strip packing problem, pallet loading problem, vehicle loading problem, container loading problem, layout problem, partitioning problem*, etc.

Kind of assignment: (i) all large items and a selection of small items, and (ii) a selection of large items and all small items.

Assortment of large items: (i) one item, (ii) many identical items, and (iii) many different items.

Assortment of small items: (i) many different items, (ii) many items of relatively few different figures, and (iii) many identical items.

Dimensionality: (i) one dimensional, (ii) two dimensional, (iii) three dimensional, and (iv) n-dimensional with $n > 3$.

Shapes (e.g., two dimensional): (i) rectangle, (ii) polygon, (iii) circle, and (iv) irregular.

For example, the knapsack problem requires to assign a selection of small items to one large item, and its dimensionality is one. Another example is the two dimensional bin packing problem which requires to assign all small items to a selection of large items, and its dimensionality is two. The detailed classification of cutting and packing problems is summarized in [52][23][14][24].

The *one dimensional cutting stock problem* (1D-CSP) is one of the most representative cutting problems, and it has been intensively studied since 1960s [59][76]. This problem arises in many industries such as steel, paper, wood, glass and fiber. Due to its wide variety of materials to handle, and industries, the 1D-CSP has been stated in many models. In 1D-CSP, we are given a sufficient number of stock rolls of the same length L, and m products $M = \{1, 2, \ldots, m\}$ of given lengths $(l_1, l_2, \ldots, l_m)$, whose demands are $(d_1, d_2, \ldots, d_m)$.

A standard formulation of 1D-CSP is to describe it in terms of the variables associated with *cutting patterns*, where a cutting pattern (or pattern) is a feasible combination of products cut from one stock roll. A cutting pattern is described as $p_j = (a_{1j}, a_{2j}, \ldots, a_{mj})$, where

$a_{ij} \in \boldsymbol{Z}_+$ (the set of nonnegative integers) is the number of products i cut from the cutting pattern p_j. We say a cutting pattern p_j satisfying

$$\sum_{i \in M} a_{ij} l_i \leq L \tag{1.3}$$

feasible, and let S denote the set of all feasible cutting patterns. It is often necessary in practice to consider additional constraints on cutting patterns. One of the common constraints is that the residual length of a cutting pattern should be smaller than that of the smallest product, i.e.,

$$L - \sum_{i \in M} a_{ij} l_i < \min_{i \in M} l_i. \tag{1.4}$$

A cutting pattern satisfying (1.4) is called *complete-cut* cutting pattern [83]. A solution of 1D-CSP consists of a set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_{|\Pi|}\} \subseteq S$, and the corresponding numbers of their applications $X = \{x_1, x_2, \ldots, x_{|\Pi|}\} \in \boldsymbol{Z}_+^{|\Pi|}$ (i.e., the number of times the cutting pattern is used). A typical cost function is the amount of residual pieces of the used stock rolls, called *trim loss*, which are usually treated as waste product. The problem of minimizing the trim loss (i.e., minimizing the number of required stock rolls) has been intensively studied as the *standard 1D-CSP*. This problem asks a solution (Π, X) which minimizes the number of required stock rolls while satisfying the demands of all products, and is formulated as follows:

$$\begin{aligned} \text{(1D-CSP)} \quad \text{minimize} \quad & f(\Pi, X) = \sum_{p_j \in \Pi} x_j \\ \text{subject to} \quad & \sum_{p_j \in \Pi} a_{ij} x_j \geq d_i \ \text{ for all } i \in M \\ & \Pi \subseteq S \\ & x_j \in \boldsymbol{Z}_+ \ \text{ for all } p_j \in \Pi. \end{aligned} \tag{1.5}$$

A sample of 1D-CSP is illustrated in Figure 1.1. As the standard 1D-CSP contains the bin packing problem (BPP), which is known to be strongly NP-hard [31], as a special case, the standard 1D-CSP is clearly a hard problem. The standard 1D-CSP has been studied since 1960s, as a variant of integer linear programming problem (ILP), which has a huge number of columns corresponding to all feasible cutting patterns. The number of all feasible cutting patterns is roughly estimated as $O(m^k)$ where k represents the average number of products in a cutting pattern. Pierce [76] showed that the number of cutting patterns can easily run

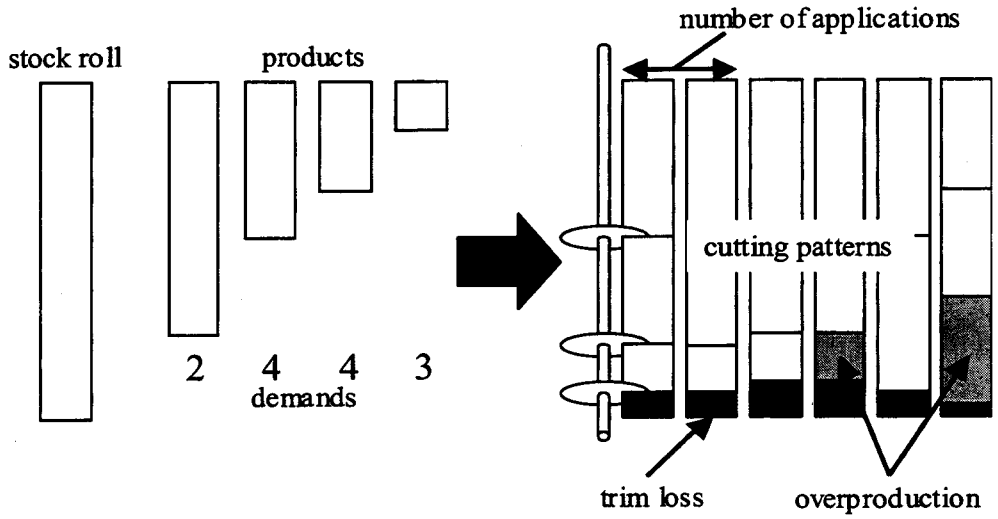

Figure 1.1: A sample of one dimensional cutting stock problem

into millions in practice situations when the average length of products is relatively small to that of stock rolls.

A classical approach to the standard 1D-CSP is based on an optimal solution of its linear programming (LP) relaxation. As it is impractical to consider all feasible cutting patterns, Gilmore and Gomory [34][35] proposed an ingenious *column generation method*, which determines the next cutting pattern necessary to improve the current solution (Π, X) by solving the associated knapsack problem. This made it possible to solve the standard 1D-CSP by linear programming without enumerating all feasible cutting patterns, and consequently they solved the standard 1D-CSP in much less time than would be required if all feasible cutting patterns were used.

We first give the problem of computing an optimal $X = \{x_1, x_2, \ldots, x_n\}$ for a given set of n cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$, and it is formulated as an integer linear programming problem (ILP):

$$\begin{aligned} (\mathrm{ILP}(\Pi)) \quad \text{minimize} \quad & f(\Pi, X) = \sum_{j=1}^{n} x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq d_i \quad \text{for } i = 1, \ldots, m \\ & x_j \in \boldsymbol{Z}_+ \quad \text{for } j = 1, \ldots, n. \end{aligned} \tag{1.6}$$

Now we consider the LP relaxation of (1.6) by replacing the integer constraints $x_j \in \boldsymbol{Z}_+$ with $x_j \geq 0$ for $j = 1, \ldots, n$. The LP relaxation is stated as the following linear programming problem (LP):

$$\begin{aligned} (\mathrm{LP}(\Pi)) \quad \text{minimize} \quad & \bar{f}(\Pi, X) = \sum_{j=1}^{n} x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq d_i \ \text{ for } i = 1, \ldots, m \\ & x_j \geq 0 \ \text{ for } j = 1, \ldots, n. \end{aligned} \tag{1.7}$$

Let y_i be the dual variable associated with the i-th constraint of LP(Π). The dual problem of LP(Π) can be formulated as

$$\begin{aligned} (\mathrm{DLP}(\Pi)) \quad \text{maximize} \quad & \sum_{i=1}^{m} d_i y_i \\ \text{subject to} \quad & \sum_{i=1}^{m} a_{ij} y_i \leq 1 \ \text{ for } j = 1, \ldots, n \\ & y_i \geq 0 \ \text{ for } i = 1, \ldots, m. \end{aligned} \tag{1.8}$$

An optimal solution $\overline{Y} = \{\overline{y}_1, \overline{y}_2, \ldots, \overline{y}_m\}$ of DLP(Π) gives us the means for determining the next cutting pattern to enter the basis of LP(Π). Now we generate the next cutting pattern $p' = (a'_1, a'_2, \ldots, a'_m)$ by solving the following knapsack problem:

$$\begin{aligned} (\mathrm{KNAP}(\Pi)) \quad \text{minimize} \quad & z(\Pi, p') = \sum_{i=1}^{m} \overline{y}_i a'_i \\ \text{subject to} \quad & \sum_{i=1}^{m} l_i a'_i \leq L \\ & a'_i \in \boldsymbol{Z}_+ \ \text{ for } i = 1, \ldots, m. \end{aligned} \tag{1.9}$$

If $z(\Pi, p') \leq 1$ holds, no feasible cutting pattern $p' \in S$ can improve the current solution $(\Pi, \overline{X})$ of LP(Π); otherwise the new cutting pattern p' satisfying $z(\Pi, p') > 1$ can be used to the next cutting pattern. Figure 1.3 illustrates the improvement of $\bar{f}$ induced by a new cutting pattern p' satisfying $z(\Pi, p') > 1$. The optimal value of the LP relaxation LP(Π) is equivalent to that of its dual problem DLP(Π) due to the duality theorem. As the new plane corresponding to a new cutting pattern p' satisfying $z(\Pi, p') > 1$ necessarily cuts the feasible space of dual problem DLP(Π), the optimal value of the LP relaxation is improved by the new cutting pattern p'. Gilmore and Gomory's algorithm starts from a feasible solution (Π, X) of

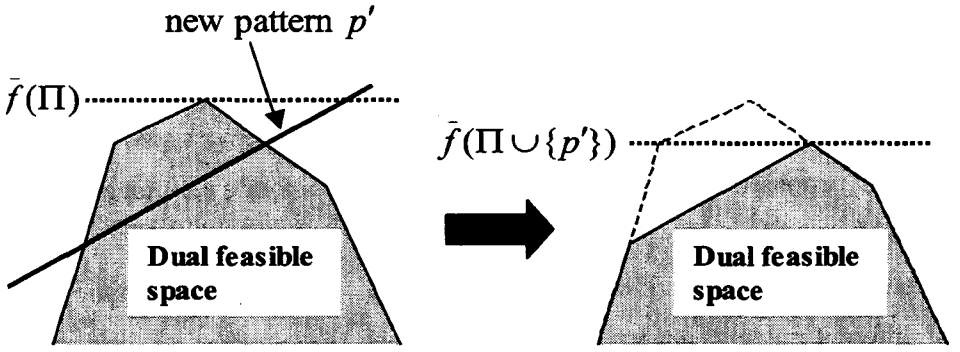

Figure 1.2: The improvement of the current solution induced by a new cutting pattern

the standard 1D-CSP, and repeatedly adds a next cutting pattern p' satisfying $z(\Pi, p') > 1$ by solving KNAP(Π) until no cutting pattern p' satisfying $z(\Pi, p') > 1$ is found.

Once an LP optimal solution is found, it can be modified in a number of ways to obtain integer values $X = \{x_1, x_2, \ldots, x_{|\Pi|}\}$ satisfying all demands. Marcotte [68] proved that certain classes of 1D-CSP have the *integer round-up property* (IRUP), which implies that the difference between the optimal value of the original 1D-CSP and that of its LP relaxation is small. Scheithauer and Terno [79][81] investigated the modified integer round-up property (MIRUP) that the optimal value is not greater than that of the corresponding LP relaxation rounded up plus one in a certain class of the standard 1D-CSP. They showed that the standard 1D-CSP has subproblems with property MIRUP. Wäscher and Gau [98] conducted a large number of numerical computations to compare several rounding heuristics for the standard 1D-CSP. They observed that these heuristics obtained the optimal values in most instances, and the differences between the values of these heuristics and the optimal values are rather small for other instances. Based on Gilmore and Gomory's column generation method and the integer round-up properties, several branch-and-bound algorithms have been developed with certain computational success [80][93][94][18].

1.4 Previous Works on the Pattern Minimization in 1D-CSP

In recent years, the costs of other factors than the trim loss have become more dominant. Among them is the setup costs for changing cutting patterns, and it is impractical to use many different cutting patterns. Although the branch-and-bound algorithms based on column

generation could solve the standard 1D-CSP in the practical sense, their solutions are not desirable in real applications since the number of different cutting pattern in their solutions may become very close to the number of products. Hence, several types of heuristic algorithms have been developed to reduce the number of different cutting patterns in 1D-CSP.

Haessler [48] proposed a modification of Gilmore and Gomory's column generation method. He restricted the number of product a'_i in the next cutting pattern $p' = (a'_1, a'_2, \ldots, a'_n)$ to increase the number of its applications x'. His idea is based on an observation that the number of different cutting pattern is often small when the numbers of their applications are relatively large. However, he could not achieve effective reductions on the number of different cutting patterns in his computational results.

Walker [97] introduced the fixed charge into LP formulation of 1D-CSP for the setup of cutting patterns, and considered the *fixed charge problem* (FCP) as follows:

$$\begin{aligned} \text{(FCP)} \quad \text{minimize} \quad & \sum_{p_j \in S} (c_j x_j + f_j v_j) \\ \text{subject to} \quad & \sum_{p_j \in S} a_{ij} x_j = d_i \ \text{ for all } i \in M \\ & x_j \geq 0 \ \text{ for all } p_j \in S \\ & v_j = \begin{cases} 1 & x_j > 0 \\ 0 & x_j = 0 \end{cases} \ \text{ for all } p_j \in S, \end{aligned} \tag{1.10}$$

where f_j represents the fixed cost to cutting pattern p_j. He proposed a two phase heuristic algorithm SWIFT which is a variant of the simplex algorithm. SWIFT utilizes an initial solution found by the conventional simplex algorithm, and then selects a new cutting pattern to enter the basis only if the additional setup cost is less than compensated for by the reduction in the number of used stock rolls. After having a locally optimal solution in this manner (i.e., no solution satisfying the above condition is found by changing one cutting pattern in the basis), then SWIFT searches an improving solution by exhaustively attempting to swap basic cutting patterns with non-basic cutting patterns; it first applies single swaps, and at the last phase applies double swaps. Farley and Richardson [26] proposed an improvement of SWIFT to reduce the total setup cost of cutting patterns in 1D-CSP. The algorithm uses additional pivoting rules not to increase the number of basic variables corresponding to cutting patterns. Their computational results unfortunately showed that the total number of required stock rolls increased rapidly as the number of different cutting patterns is reduced.

Haessler proposed a pattern generating heuristic algorithm called the *sequential heuristic*

procedure (SHP) [46][47][49]. SHP starts from an empty set of cutting patterns, and repeatedly adds a new cutting pattern to the current solution until all demands are satisfied, where the next cutting patterns should have small trim loss and large number of applications. In each step, SHP first computes the following parameters:

MAXTL: The maximum allowable trim loss

MINR (resp., MAXR): The minimum (resp., maximum) numbers of products permitted in the next cutting pattern

MINU: The minimum number of applications to the next cutting patterns

SHP searches a next cutting pattern exhaustively among those satisfying all constraints given by the above parameters. If a desirable cutting pattern is found, SHP applies it at maximum possible times, under the constraint that the residual demands for all products are not exceeded, i.e., $\sum_{p_j \in \Pi} a_{ij} x_j \leq d_i$. If no cutting pattern is found, SHP decreases the parameter MINU to the next cutting pattern and searches again. Vahrenkamp [92] proposed a variant of SHP, in which a new cutting pattern is generated by a simple randomized algorithm. Gradisar [44] applied SHP to a variant of 1D-CSP in which stock rolls may have different lengths. Sweeney [84] proposed a hybrid algorithm based on SHP and linear programming (LP). SHP and its variants can be used effectively to solve 1D-CSP when the lengths of the products are small relative to that of the stock roll. Unfortunately, it is observed that they do not work well when the lengths of the products are large relative to that of the stock roll.

Johnston [58] and Goulimis [43] proposed a *pattern combination heuristic algorithm.* The algorithm starts from a solution obtained by another algorithm designed to minimize the number of used stock rolls (e.g., solve the standard 1D-CSP), and reduces the number of different cutting patterns by combining cutting patterns together; i.e., it select a number of cutting patterns in the solution, and replaces them with a smaller number of new cutting patterns such that the amount of products covered by the new cutting patterns is equivalent to that covered by the removed cutting patterns. Johnston proposed a simple algorithm for deciding whether two cutting patterns can be combined into one cutting pattern. He also mentioned but did not fully describe an tree-search algorithm for combining three cutting patterns into two cutting patterns. It repeats these two algorithms until no further reduction can be made. Foerster and Wäscher [29] developed an algorithm based on the pattern

combination heuristics, called KOMBI, which applies many types of combination heuristics in addition to Johnston's heuristics.

Vanderbeck [95] considered a formulation of 1D-CSP which minimizes the number of different cutting patterns while using a given number of stock rolls or less, and it was called the *pattern minimization problem* (1D-PMP).

$$\begin{aligned} \text{(1D-PMP)} \quad \text{minimize} \quad & |\Pi| \\ \text{subject to} \quad & \sum_{p_j \in \Pi} a_{ij} x_j \geq d_i \ \text{ for all } i \in M \\ & \sum_{p_j \in \Pi} x_j \leq f_{UB} \\ & \Pi \subseteq S \\ & x_j \in \boldsymbol{Z}_+ \ \text{ for all } p_j \in \Pi, \end{aligned} \tag{1.11}$$

where f_{UB} is the upper bound on the number of used stock rolls, which is an input parameter given by users. As 1D-PMP contains the bin packing problem (BPP) as a special case that all demands d_i are set to 1, 1D-PMP is also strongly NP-hard. McDiarmid [67] considered a special case of 1D-PMP where any two products fit on a stock roll ($l_i + l_j \leq L, \forall i, j$) but no three do ($l_i + l_j + l_k > L, \forall i, j, k$). Although the minimum number of used stock rolls is known to be $\lceil \sum_{i \in M} d_i l_i / 2 \rceil$, he showed that this special case of 1D-PMP is still strongly NP-hard. Vanderbeck proposed an exact algorithm for 1D-PMP. He first described 1D-PMP as an integer quadratic programming problem (IQP), which is then decomposed into a number of auxiliary integer linear programming problems (ILP) with strong LP relaxations. Then a branch-and-cut algorithm is applied while using a column generation method. According to his computational results, this algorithm could solve many small instances exactly, but failed to obtain optimal solutions for several instances of moderate sizes in two hours.

1.5 Research Objectives and Overview of the Thesis

In this thesis, we take a new approach by considering the number of different cutting patterns n as an input parameter given by users. We call this variant of 1D-CSP, as the *pattern restricted* version of 1D-CSP (1D-PRP), which minimizes the number of stock rolls while using n different cutting patterns or less.

$$\text{(1D-PRP)} \quad \text{minimize} \quad f(\Pi, X) = \sum_{p_j \in \Pi} x_j \tag{1.12}$$

$$\text{subject to} \quad \sum_{p_j \in \Pi} a_{ij} x_j \geq d_i \ \text{ for all } i \in M$$
$$\Pi \subseteq S$$
$$|\Pi| \leq n$$
$$x_j \in \boldsymbol{Z}_+ \ \text{ for all } p_j \in \Pi,$$

where n is an input parameter set by users. We suppose that the number of different cutting patterns n is less than that of products m. As 1D-PRP is a simple extension of the standard 1D-CSP that is obtained by only adding the constraint $|\Pi| \leq n$ to the standard 1D-CSP, 1D-PRP is also strongly NP-hard. This new approach is similar to the fixed-k approach to the graph coloring problem (GCP) [55][12].

In general, it becomes easier to find a solution using a smaller number of stock rolls as the number of different cutting patterns becomes larger. In this sense, there is a trade-off between the number of required stock rolls and the number of different cutting patterns. By solving 1D-PRP for different parameter values n, we can obtain a trade-off curve as illustrated in Figure 1.3. Using this we can make a more careful analysis of the trade-off between two objective functions: the number of required stock rolls f and the number of different cutting patterns n. It is also possible to solve the pattern minimization problem (1D-PMP) by searching the minimum feasible n (i.e., using f_{UB} stock rolls or less) by employing binary search, for example, over the space n.

In this thesis, we focus on this new formulation of 1D-CSP, and propose new approximate algorithms based on local search algorithm (LS). A solution of 1D-PRP consists of a set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$ and the corresponding numbers of their applications $X = \{x_1, x_2, \ldots, x_n\}$ (i.e., x_j means the number of times the cutting pattern p_j used). Our local search algorithms to 1D-PRP start from an initial feasible solution (Π^{init}, X^{init}) obtained by a certain heuristic algorithm. Solutions in the neighborhood of the current solution (Π, X) are generated by perturbating one or two cutting patterns in the current set of cutting patterns Π. We give natural definitions of two neighborhoods $N_1(\Pi)$ and $N_2(\Pi)$ for our local search algorithms as follows:

$$N_1(\Pi) = \{\Pi \cup \{p'_j\} \setminus \{p_j\} \mid p_j \in \Pi, p'_j \in S \setminus \Pi\}, \tag{1.13}$$
$$N_2(\Pi) = \{\Pi \cup \{p'_{j_1}, p'_{j_2}\} \setminus \{p_{j_1}, p_{j_2}\} \mid p_{j_1}, p_{j_2} \in \Pi, p'_{j_1}, p'_{j_2} \in S \setminus \Pi\}, \tag{1.14}$$

where S is the set of all feasible cutting patterns. For each neighbor $\Pi' \in N_1(\Pi)$ (resp., $N_2(\Pi)$), the numbers of applications X' to the cutting patterns Π' are computed by solving

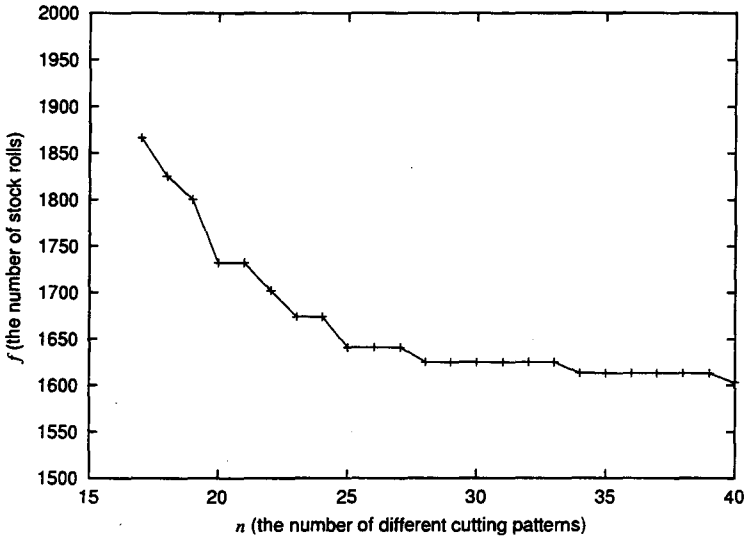

Figure 1.3: Trade-off curve between the number of stock rolls f and the number of different cutting patterns n

the auxiliary integer linear programming problem (ILP) given in (1.6). A naive local search algorithm to 1D-PRP is described as follows:

Algorithm Naive Local Search

Input: Lengths l_i and demands d_i of all products $i \in M$, the number of different cutting patterns n, and the length of stock rolls L.

Output: A set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$ and the numbers of their applications $X = \{x_1, x_2, \ldots, x_n\}$ or "failure".

Step 1: Apply a heuristic algorithm to obtain an initial feasible solution (Π^{init}, X^{init}). If no feasible solution is found, output "failure" and halt; otherwise set $(\Pi, X) := (\Pi^{init}, X^{init})$.

Step 2: If there is a feasible solution $\Pi' \in N_1(\Pi)$ (or $N_2(\Pi)$) such that $f(\Pi', X') < f(\Pi, X)$ holds, set $(\Pi, X) := (\Pi', X')$ and return to Step 2. Otherwise (i.e., $f(\Pi', X') \geq f(\Pi, X)$ holds for all $\Pi' \in N_1(\Pi)$ (or $N_2(\Pi)$)) output (Π, X) and halt.

Although local search and metaheuristic algorithms have many good features, they are

not straightforwardly applicable to 1D-PRP. The intractability of 1D-PRP arises from the following facts that:

Initial feasible solution: As all products $i \in M$ must appear at least one cutting pattern $p_j \in \Pi$, the problem of finding a feasible solution to 1D-PRP is formulated as the bin packing problem (BPP) known to be NP-hard,

Neighborhood: As the number of all feasible cutting patterns $|S|$ is roughly estimated as $O(m^k)$, where k represents the average number of products in a cutting pattern. It can grow exponentially in the number of products m. It is not realistic to consider all possible feasible cutting patterns,

Auxiliary integer programming problem: The problem ILP(Π) given in (1.6) is strongly NP-hard since it contains the set covering problem (SCP) as a special case that all a_{ij}, d_i and x_j are set to 1,

and it necessitates to compute them much more efficiently. To overcome these difficulties, we propose two local search algorithms in Chapter 2 and 3, respectively.

The first is a local search algorithm based on an adaptive pattern generation (LS-APG). It starts from an initial solution obtained by a modified first fit heuristic (MFF) known for the bin packing problem (BPP). Solutions in the neighborhood are constructed by removing two cutting patterns in the current solution and adding two new cutting patterns. To restrict the neighbor solutions, we develop a heuristic algorithm that tries to generate only promising new cutting patterns, called the *adaptive pattern generation* (APG). The adaptive pattern generation constructs new cutting patterns based on residual demands when two cutting patterns are removed. The numbers of applications to the cutting patterns are computed by a heuristic algorithm, which is based on an optimal solution of the LP relaxation (1.7) of the auxiliary integer linear programming problem (1.6).

The second is a local search algorithm based on linear programming techniques (LS-LP). It starts from an initial solution obtained by a modified first fit heuristic (FF) other than that of LS-APG. Solutions in the neighborhood are restricted to those obtainable by perturbating one cutting pattern in the current set of cutting patterns. In order to find promising directions, we utilize a dual optimal solution of the LP relaxation (1.7). Although the primal solution of the LP relaxation is not integer valued, it provides reasonably accurate information as the integrality gap is rather small in most instances of 1D-CSP. Since the local

search algorithm requires to solve a large number of LP relaxations which are only slightly different each other, we start the simplex algorithm from the optimal simplex tableau of the previous solution, instead of starting it from scratch. We modify the simplex algorithm by applying the sensitivity analysis techniques, and apply a variant of the simplex algorithm called the *criss-cross algorithm* [103][85] to compute an optimal solution.

We also consider another formulation of 1D-CSP based on a real application in a chemical fiber industry. We give a variant of 1D-PRP called the *quadratic deviation minimization problem* (1D-QDP) which minimizes the amount of quadratic deviation from all demands while using a given number of different cutting patterns n:

$$\begin{aligned} \text{(1D-QDP)} \quad \text{minimize} \quad & f(\Pi, X) = \sum_{i \in M} \left(\sum_{p_j \in \Pi} a_{ij} x_j - d_i \right)^2 \\ \text{subject to} \quad & \Pi \subseteq S \\ & |\Pi| \leq n \\ & x_j \in \boldsymbol{Z}_+ \ \text{ for all } p_j \in \Pi, \end{aligned} \tag{1.15}$$

where S is the set of all feasible cutting patterns. Although the above formulation ignores the trim loss, if necessary, we can control the quality of trim loss to some extent by applying appropriate constraints on cutting patterns, e.g., restricting S to be the set of complete-cut patterns defined in (1.4). In this formulation, we allow both overproduction and shortage because the additional cost due to the shortage is relatively small in some applications such as the chemical fiber industry. For this problem, we propose a local search algorithm based on the *quadratic version of the adaptive pattern generation* (LS-QAPG). Solutions in the neighborhood are generated by removing one cutting pattern in the current solution and adding one new cutting pattern. To compute the numbers of applications to the cutting patterns, we propose a heuristic algorithm based on the *nonlinear Gauss-Seidel method* [7]. As it is not realistic to consider all possible feasible cutting patterns, we restrict the candidate cutting patterns to those generated by the quadratic version of adaptive pattern generation, where we also indirectly reducing the trim loss by restricting candidate cutting patterns to those having small trim loss.

The thesis is organized as follows. In Chapter 2, a local search algorithm based on an adaptive pattern generation (LS-APG) is proposed. In Chapter 3, a local search algorithm based on linear programming techniques (LS-LP) is proposed. In Chapter 4, we propose a local search algorithm (LS-QAPG) to a variant of 1D-PRP given in (1.15) called 1D-QDP,

which comes from a real application of a chemical fiber industry. Finally, in Chapter 5, we summarize our study in the thesis.

Chapter 2

A Local Search Algorithm Based on Adaptive Pattern Generation

2.1 Introduction

In this chapter, we propose a local search algorithm for 1D-PRP based on an adaptive pattern generation, in which neighbor solutions are constructed by removing two cutting patterns and adding two new cutting patterns in the current solution. This algorithm is named the local search algorithm based on adaptive pattern generation (LS-APG). As mentioned in Section 1.5, we still have to specify some details before implementing local search algorithms to 1D-PRP, i.e., (i) how to construct an initial feasible solution, (ii) how to compute auxiliary integer linear programming problems (ILP) efficiently, and (iii) how to find promising solutions sufficiently among all neighbor solutions in the neighborhood. Our local search algorithm in this chapter constructs an initial feasible solution by applying a variant of the first-fit heuristic algorithm (FF) known for the bin packing problem (BPP) [31], solves the auxiliary ILP by a heuristic algorithm based on its LP relaxations, and generates candidate cutting patterns by applying an adaptive pattern generation (APG) which is based on residual demands when two cutting patterns are removed from the current solution. Finally, we conduct computational experiment for randomly generated problem instances, and observe that LS-APG obtains a wide variety of good solutions comparable to SHP and KOMBI, and it provides useful trade-off curves between the number of different cutting patterns and the number of required stock rolls.

2.2 Generation of an Initial Solution

If there is no restriction on the number of different cutting patterns, as in the standard 1D-CSP, it is easy to construct a feasible solution. However, just finding a feasible solution is not trivial in 1D-PRP. The problem of finding a feasible solution under a given number of cutting patterns n is equivalent to the bin packing problem (BPP) known to be NP-complete [31]. Here, we suppose that the number of different cutting patterns n is less than that of products m, because a feasible solution is easy to obtain in the case of $n \geq m$.

Bin Packing Problem (BPP)

Input: Lengths l_i of all products $i \in M$, the number of different cutting patterns n, and the length of stock rolls L.

Output: A partition of M into n disjoint subsets $M_1, M_2, \ldots, M_n \subseteq M$ subject to $\sum_{i \in M_j} l_i \leq L$ for all $M_j \subseteq M$.

From a solution of BPP, we can construct cutting patterns $p_j = (a_{1j}, a_{2j}, \ldots, a_{mj})$ and their numbers of applications x_j by setting $a_{ij} := 1$ for $i \in M_j$ (and $a_{ij} := 0$ for $i \notin M_j$) and $x_j := \max_{i \in M_j} d_i$. The resulting set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$ and the numbers of their applications $X = \{x_1, x_2, \ldots, x_n\}$ is obviously a feasible solution to 1D-PRP. We first consider how to construct a feasible solution by solving BPP, and then modify it to reduce the number of stock rolls.

To construct a feasible solution, we prepare a heuristic algorithm based on the first fit principle (FF), where FF is one of the representative approximation algorithms known for BPP. After preparing n stock rolls of lengths L, FF sequentially assigns each product i into the stock roll with the lowest index among these having the residual capacity of at least l_i. However, to guarantee that every stock roll is assigned at least one product, and to improve the quality as an initial solution of the local search algorithm, we modify FF as follows and call this algorithm as the modified first fit algorithm (MFF). First, we sort all product $i \in M$ in the descending order of demands d_i, where $\sigma(k)$ denotes the k-th product in the resulting order. We assign all products to stock rolls in this order. We also define an aspiration length

$$L' = \alpha L, \tag{2.1}$$

where α is a program parameter satisfying $0 < \alpha \leq 1$. If the processed length of the current stock roll exceeds L' after product $\sigma(k)$ is assigned, the next product $\sigma(k+1)$ is assigned to

the next stock roll. This algorithm assigns at least one product to every stock roll if

$$\alpha = \frac{\sum_{i \in M} l_i}{nL} \tag{2.2}$$

is used, and it is equivalent to the original FF if α is set to 1. If MFF terminates before m products are assigned into n stock rolls, we conclude the failure of MFF.

Algorithm MFF

Input: Lengths l_i and demands d_i of products $i \in M$, the number of different cutting patterns n, the length of stock rolls L and a given program parameter α.

Output: n disjoint subsets $M_1, M_2, \ldots, M_n \subseteq M$ or "failure".

Step 1: Set $M_j := \emptyset$ for $j = 1, 2, \ldots, n$, and $L' := \alpha L$.

Step 2: Sort all products $i \in M$ in the descending order of d_i, where $\sigma(k)$ denotes the k-th product in this order. Set $k := 1$ and $j := 1$.

Step 3: If $l_{\sigma(k)} \leq L - \sum_{i \in M_j} l_i$ and $\sum_{i \in M_j} l_i \leq L'$ hold, set $M_j := M_j \cup \{\sigma(k)\}$, $k := k + 1$ and $j := 1$; otherwise set $j := j + 1$. If $k \leq m$ and $j \leq n$ hold, return to Step 3; otherwise go to Step 4.

Step 4: If $k > m$ holds, output $M_1, M_2, \ldots, M_n$ and halt; otherwise output "failure" and halt.

In LS-APG, we first try MFF with $\alpha = \sum_{i \in M} l_i / nL$. If MFF fails to obtain a feasible solution, we switch to the first fit decreasing heuristic (FFD); i.e., it sorts all products in the descending order of l_i (not d_i) in Step 2, and uses $\alpha = 1$ in Step 1. The other part of algorithm does not change. If this attempt also fails, we conclude the failure of LS-APG.

Now assume that MFF outputs n disjoint subsets $M_1, M_2, \ldots, M_n \subseteq M$. For each M_j, we consider the construction of cutting patterns $p_j = (a_{ij} \mid i \in M_j)$ and the number of its applications x_j so that the demands $d_i (i \in M_j)$ are met with the smallest x_j (nonnegative real number). The problem of generating a cutting pattern p_j from a given M_j is described as follows, where x_j and a_{ij} are both variables.

$$\begin{aligned} (\text{1D-CSP}_1(j)) \quad & \text{minimize} \quad x_j \\ & \text{subject to} \quad a_{ij} x_j \geq d_i \ \text{ for all } i \in M_j \end{aligned} \tag{2.3}$$

$$\sum_{i \in M_j} a_{ij} l_i \leq L$$
$$a_{ij} \in \boldsymbol{Z}_+ \text{ for all } i \in M_j$$
$$x_j \geq 0.$$

For this problem, we propose an exact algorithm called the *single adaptive pattern generation* (SAPG). It starts from $a_{ij} := 1$ for all $i \in M_j$ and $x_j := \max_{i \in M_j} d_i$, and gradually increases a_{ij} to reduce the number of applications x_j. Here, we say that product i is *bottleneck* if $a_{ij} x_j = d_i$ holds for the current a_{ij} and x_j. Let $L^{res} = L - \sum_{i \in M_j} a_{ij} l_i$ denote the residual length of the stock roll. If $l_i \leq L^{res}$ holds for a bottleneck product i, we have to increase a_{ij} (by one) in order to reduce x_j. We repeat this process as long as it is possible.

Algorithm SAPG

Input: A set of products $M_j \subseteq M$, demands d_i and lengths l_i of all products $i \in M_j$, and the length of the stock roll L (M_j satisfies $\sum_{i \in M_j} l_i \leq L$).

Output: A feasible cutting pattern $p_j = (a_{ij} \mid i \in M_j)$ and the number of its applications x_j.

Step 1: Set $a_{ij} := 1$ for all $i \in M_j$, $x_j := \max_{i \in M_j} d_i$ and $L^{res} := L - \sum_{i \in M_j} l_i$.

Step 2: If there is no bottleneck product $i \in M_j$ such that $l_i \leq L^{res}$ holds, output $p_j = (a_{ij} \mid i \in M_j)$ and x_j, and halt; otherwise take a bottleneck product $i \in M_j$ (i.e., $a_{ij} x_j = d_i$ holds) such that $l_i \leq L^{res}$ holds, set $a_{ij} := a_{ij} + 1$, $x_j := \max_{i \in M_j} \frac{d_i}{a_{ij}}$, $L^{res} := L^{res} - l_i$ and return to Step 2.

Here, we prove the following theorem about SAPG.

Theorem 1 *SAPG outputs an optimal solution of problem 1D-CSP*$_1(j)$.

Proof. Let x_j and $p_j = (a_{ij} \mid i \in M_j)$ be the output of SAPG. Since a_{ij} increases only when $a_{ij} x_j = d_i$ holds, and variable x_j is non-increasing during the execution of SAPG, we have

$$\frac{d_i}{a_{ij} - 1} > x_j \text{ for all } a_{ij} \geq 2. \tag{2.4}$$

Now let x_j^* and $p_j^* = (a_{ij}^* \mid i \in M_j)$ be an optimal solution of 1D-CSP$_1(j)$. If there is an a_{ij}^* such that $a_{ij}^* < a_{ij}$, it follows

$$x_j^* \geq \frac{d_i}{a_{ij}^*} \geq \frac{d_i}{a_{ij} - 1} > x_j \tag{2.5}$$

which contradicts to the optimality of x_j^*. Therefore, $a_{ij}^* \geq a_{ij}$ holds for all $i \in M_j$. Now if $x_j^* < x_j$ holds (i.e., x_j is not optimal), there is a bottleneck product i for which $a_{i'j}^* > a_{i'j}$ holds. But this implies

$$0 \leq L - \sum_{i \in M_j} a_{ij} l_i - l_{i'} = L^{res} - l_{i'}, \tag{2.6}$$

and $a_{i'j}$ would have been increased in Step 2 of SAPG. This is a contradiction, and shows that $x_j^* = x_j$ holds. □

Finally, we summarize the algorithm MFF_INIT that constructs an initial feasible solution using MFF and SAPG for $j = 1, 2, \ldots, n$.

Algorithm MFF_INIT

Input: Lengths l_i and demands d_i of products $i \in M$, the number of different cutting patterns n, and the length of stock rolls L.

Output: A set of n cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$ and the numbers of their applications $X = \{x_1, x_2, \ldots, x_n\}$, or "failure".

Step 1: Apply MFF to obtain n disjoint subsets $M_1, M_2, \ldots, M_n \subseteq M$. If this attempt fails, apply the FFD version of MFF. If FFD still outputs "failure", we output "failure" and halt.

Step 2: For $j = 1, 2, \ldots, n$, compute cutting pattern p_j and the number of its applications x_j by applying SAPG. Output the set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$ and the numbers of their applications $X = \{x_1, x_2, \ldots, x_n\}$.

Note that the numbers of applications x_j computed in MFF_INIT are real numbers, and it is necessary to round them into integers before starting the local search algorithm. This rounding process will be explained in the next section for any set of patterns Π (not restricted to the initial set obtained by MFF_INIT).

2.3 Solving Auxiliary Integer Linear Programming Problems

For a given set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$, the problem of computing the numbers of their applications $X = \{x_1, x_2, \ldots, x_n\}$ to minimize the cost function $f(\Pi, X)$

can be formulated as the following integer linear programming problem ILP(Π):

$$\begin{aligned} (\mathrm{ILP}(\Pi)) \quad \text{minimize} \quad & f(\Pi, X) = \sum_{j=1}^{n} x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq d_i \ \text{ for } i = 1, 2, \ldots, m \\ & x_j \in \boldsymbol{Z}_+ \ \text{ for } j = 1, 2, \ldots, n. \end{aligned} \tag{2.7}$$

As it contains the set covering problem (SCP), known to be NP-hard [31], as a special case, we consider to find an approximate solution $\widehat{X} = \{\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n\}$ and its cost value $f(\Pi, \widehat{X})$ by a heuristic algorithm. Our heuristic algorithm SOLVE_IP first solves the LP relaxation LP(Π) of ILP(Π):

$$\begin{aligned} (\mathrm{LP}(\Pi)) \quad \text{minimize} \quad & \bar{f}(\Pi, X) = \sum_{j=1}^{n} x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq d_i \ \text{ for } i = 1, 2, \ldots, m \\ & x_j \geq 0 \ \text{ for } j = 1, 2, \ldots, n. \end{aligned} \tag{2.8}$$

Let $\overline{X} = \{\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\}$ denote an optimal solution of the LP relaxation LP(Π). SOLVE_IP starts from $\hat{x}_j := \lfloor \overline{x}_j \rfloor$ for all $j = 1, 2, \ldots, n$. In order to obtain an integer feasible solution, it first sorts variables $\hat{x}_j$ in the descending order of fractions $\overline{x}_j - \lfloor \overline{x}_j \rfloor$, and then rounds them up in the resulting order until all demands are satisfied.

Algorithm SOLVE_IP

Input: Demands d_i of all products $i \in M$, and a set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$.

Output: The numbers of applications $\widehat{X} = \{\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n\}$, or "failure".

Step 1: Compute an optimal solution $\overline{X} = \{\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\}$ of the LP relaxation LP(Π). If LP(Π) is infeasible, output "failure" and halt.

Step 2: Set $\hat{x}_j := \lfloor \overline{x}_j \rfloor$ for all $j = 1, 2, \ldots, n$.

Step 3: Sort all variables $\hat{x}_j$ in the descending order of fraction $\overline{x}_j - \lfloor \overline{x}_j \rfloor$, and let $\sigma(k)$ denote the k-th variable in this order. Set $k := 1$.

Step 4: If all demands are satisfied (i.e., $\sum_{j=1}^{n} a_{ij} \hat{x}_j \geq d_i$ holds for all $i \in M$), then output $\widehat{X} = \{\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n\}$ and halt. Otherwise if there is at least

one $i \in M$ such that $\sum_{j=1}^{n} a_{ij}\hat{x}_j < d_i$ and $a_{i\sigma(k)} > 0$ hold, then set $\hat{x}_{\sigma(k)} := \lceil \hat{x}_{\sigma(k)} \rceil$. Set $k := k + 1$ and return to Step 4.

In general, we employ the *revised simplex algorithm* [17] to solve the LP relaxation LP(Π) in the execution of SOLVE_IP. However, note that we still have a large number of LP relaxations to solve in even one execution of the local search algorithm, and it consumes much computational time.

2.4 Construction of the Neighborhood

A natural definitions of the neighborhood may be:

$$N_1(\Pi) = \{\Pi \cup \{p'_j\} \setminus \{p_j\} \mid p_j \in \Pi, p'_j \in S \setminus \Pi\}, \tag{2.9}$$

where S is the set of all feasible cutting patterns. That is, a neighbor solution $\Pi' \in N_1(\Pi)$ is constructed by removing one cutting pattern $p_j \in \Pi$, and adding one new cutting pattern $p'_j \in S \setminus \Pi$. However, according to our preliminary computational experiment, such Π' does not appear powerful enough to generate new solutions, which are substantially different from the current solution. Therefore, our local search algorithm LS-APG uses a larger neighborhood $N_2(\Pi)$ defined as follows:

$$N_2(\Pi) = \{\Pi \cup \{p'_{j_1}, p'_{j_2}\} \setminus \{p_{j_1}, p_{j_2}\} \mid p_{j_1}, p_{j_2} \in \Pi, p'_{j_1}, p'_{j_2} \in S \setminus \Pi\}. \tag{2.10}$$

As the number of feasible cutting patterns $|S|$ can grow exponentially in the number of products m, the size of $N_2(\Pi)$ may be too large to test all neighbor solutions $\Pi' \in N_2(\Pi)$, and most of them may not lead to improvement. Hence, we consider a small subset $N_2^{apg}(\Pi) \subseteq N_2(\Pi)$ hopefully containing promising solutions.

In order to explain $N_2^{apg}(\Pi)$, we define the residual demands $r_i(j_1, j_2)$ when cutting patterns p_{j_1} and p_{j_2} are removed from the current solution (Π, X):

$$r_i(j_1, j_2) = \max\left\{0, d_i - \sum_{p_j \in \Pi \setminus \{p_{j_1}, p_{j_2}\}}\right\} \text{ for } i \in M. \tag{2.11}$$

Now we introduce the following problem of constructing a pair of new cutting patterns $p'_{j_1} = (a_{ij_1} \mid i \in M(j_1, j_2))$ and $p'_{j_2} = (a_{ij_2} \mid i \in M(j_1, j_2))$, which replace the removed cutting patterns p_{j_1} and p_{j_2}, where $M(j_1, j_2) \subseteq M$ denotes the set of products $i \in M$ such that

$r_i(j_1, j_2) > 0$, where x_j and a_{ij} are both variables.

$$
\begin{aligned}
(\text{1D-CSP}_2(j_1, j_2)) \quad \text{minimize} \quad & x_{j_1} + x_{j_2} \qquad (2.12)\\
\text{subject to} \quad & a_{ij_1} x_{j_1} + a_{ij_2} x_{j_2} \geq r_i(j_1, j_2) \ \text{ for all } i \in M(j_1, j_2)\\
& \sum_{i \in M(j_1, j_2)} a_{ij_1} l_i \leq L\\
& \sum_{i \in M(j_1, j_2)} a_{ij_2} l_i \leq L\\
& a_{ij_1}, a_{ij_2} \in \boldsymbol{Z}_+ \ \text{ for all } i \in M(j_1, j_2)\\
& x_{j_1}, x_{j_2} \geq 0.
\end{aligned}
$$

However, 1D-CSP$_2(j_1, j_2)$ is NP-hard since the partition problem, known to be NP-complete, can be transformed to 1D-CSP$_2(j_1, j_2)$ by the following theorem.

Partition Problem

Input: Lengths l_i of all products $i \in M$.

Output: A subset $M' \subseteq M$ such that $\sum_{i \in M'} l_i = \sum_{M \setminus M'} l_i$ holds.

Theorem 2 *1D-CSP$_2(j_1, j_2)$ is a NP-hard problem.*

Proof. Given an instance of the partition problem, we construct an instance of 1D-CSP$_2(j_1, j_2)$ by setting $M(j_1, j_2) := M$, $r_i := 1$ for all $i \in M(j_1, j_2)$ and $L := \frac{\sum_{i \in M(j_1, j_2)} l_i}{2}$. (i) It is trivial to see that this construction of the instance of 1D-CSP$_2(j_1, j_2)$ is done in polynomial time. (ii) We claim that a solution of the partition problem can be obtained if a feasible solution of 1D-CSP$_2(j_1, j_2)$ is obtained. Let $M_1(j_1, j_2)$ and $M_2(j_1, j_2)$ be the set of products in the pattern p'_{j_1} and p'_{j_2}, respectively. We now suppose there is a feasible solution of 1D-CSP$_2(j_1, j_2)$ satisfying $\sum_{i \in M_1(j_1, j_2)} l_i < L$. As any feasible solution of 1D-CSP$_2(j_1, j_2)$ must satisfy $M_1(j_1, j_2) \cup M_2(j_1, j_2) = M(j_1, j_2)$ and $L = \frac{\sum_{i \in M(j_1, j_2)} l_i}{2}$, $\sum_{i \in M_2(j_1, j_2)} l_i > L$ holds. However, it contradicts to the feasibility of 1D-CSP$_2(j_1, j_2)$. Therefore, any feasible solution of 1D-CSP$_2(j_1, j_2)$ must satisfy $\sum_{i \in M_1(j_1, j_2)} l_i = L$ (resp., $\sum_{i \in M_2(j_1, j_2)} l_i = L$), and we can obtain a solution of the partition problem by setting $M_1 := M_1(j_1, j_2)$. From the facts (i), (ii) and the NP-completeness of the partition problem, this theorem is proved. □

Therefore, we propose a two phase heuristic algorithm called the *double adaptive pattern generation* (DAPG) to obtain a good solution of 1D-CSP$_2(j_1, j_2)$, and introduce the

neighborhood $N_2^{apg}(\Pi) \subseteq N_2(\Pi)$ using DAPG:

$$N_2^{apg}(\Pi) = \{\Pi \cup \{p'_{j_1}, p'_{j_2}\} \setminus \{p_{j_1}, p_{j_2}\} \mid p_{j_1}, p_{j_2} \in \Pi, p'_{j_1}, p'_{j_2} \in S'(j_1, j_2)\}, \tag{2.13}$$

where $S'(j_1, j_2) \subseteq S$ is the set of new cutting patterns generated by DAPG.

First, DAPG finds a partition of $M(j_1, j_2)$ into a pair of disjoint subsets $M_1(j_1, j_2)$ and $M_2(j_1, j_2)$ such that $\sum_{i \in M_1(j_1,j_2)} l_i \leq L$ and $\sum_{i \in M_2(j_1,j_2)} l_i \leq L$ hold, respectively. This part of DAPG is done by the following algorithm BALANCE-FIT. After sorting all products $i \in M(j_1, j_2)$ in the descending order of l_i, BALANCE-FIT sequentially assigns all products $i \in M(j_1, j_2)$ to the cutting pattern having longer residual length in the resulting order. Here, L_1^{res} and L_2^{res} represent the current residual lengths of the cutting patterns p'_{j_1} and p'_{j_2}, respectively.

Algorithm BALANCE-FIT

Input: A set $M(j_1, j_2) \subseteq M$, lengths l_i of all products $i \in M(j_1, j_2)$, and the length of stock rolls L.

Output: A pair of subsets $M_1(j_1, j_2), M_2(j_1, j_2) \subseteq M(j_1, j_2)$.

Step 1: Sort all products $i \in M(j_1, j_2)$ in the descending order of l_i, and let $\sigma(k)$ denote the k-th product in this order. Set $M_1(j_1, j_2) := \emptyset$, $M_2(j_1, j_2) := \emptyset$, $L_1^{res} := L$, $L_2^{res} := L$, and $k := 1$.

Step 2: If $L_1^{res} \geq L_2^{res}$ (resp., $L_1^{res} < L_2^{res}$) holds, let $h := 1$ (resp., $h := 2$). If $l_{\sigma(k)} \leq L_h^{res}$ holds, set $M_h(j_1, j_2) := M_h(j_1, j_2) \cup \{\sigma(k)\}$ and $L_h^{res} := L_h^{res} - l_{\sigma(k)}$; else output "failure" and halt.

Step 3: Set $k := k + 1$. If $k \leq |M(j_1, j_2)|$ holds, return to Step 2; otherwise output the subsets $M_1(j_1, j_2)$, $M_2(j_1, j_2)$, and halt.

Second, DAPG starts from the following feasible pair of the cutting patterns $p'_{j_1} = (a_{ij_1} \mid i \in M_1(j_1, j_2))$ and $p'_{j_2} = (a_{ij_2} \mid i \in M_2(j_1, j_2))$, which are constructed from subsets $M_1(j_1, j_2)$ and $M_2(j_1, j_2)$:

$$\begin{aligned} a_{ij_1} &:= 1 \text{ for all } i \in M_1(j_1, j_2) \\ a_{ij_2} &:= 1 \text{ for all } i \in M_2(j_1, j_2) \\ x_{j_1} &:= \max\{r_i(j_1, j_2) \mid a_{ij_1} > 0, i \in M_1(j_1, j_2)\} \\ x_{j_2} &:= \max\{r_i(j_1, j_2) \mid a_{ij_2} > 0, i \in M_2(j_1, j_2)\} \end{aligned} \tag{2.14}$$

DAPG repeats adding products to these cutting patterns in order to reduce the number of stock rolls (i.e., $x_{j_1} + x_{j_2}$) to cover the residual demands $r_i(j_1, j_2)$ for all $i \in M(j_1, j_2)$. Call a product $i \in M(j_1, j_2)$ *bottleneck* if $a_{ij_1}x_{j_1} + a_{ij_2}x_{j_2} = r_i(j_1, j_2)$ holds for the current pair of cutting patterns p'_{j_1}, p'_{j_2} and the numbers of their applications x_{j_1}, x_{j_2}. In each step, DAPG increases either a_{ij_1} or a_{ij_2} of a bottleneck product i, and correspondingly updates the numbers of applications x_{j_1}, x_{j_2}. This is repeated until the number of stock rolls and the amount of overproduction becomes unable to improve. This process is similar to that of SAPG in Section 2.2 because both algorithms repeatedly increase bottleneck products in order to reduce the number of applications. However, as there are two cutting patterns involved, DAPG has to consider the balance between them.

For the current pair of cutting patterns p'_{j_1} and p'_{j_2}, the numbers of their applications can be computed by solving the following linear programming problem $\mathrm{LP}_2(j_1, j_2)$:

$$\begin{aligned} (\mathrm{LP}_2(j_1, j_2)) \quad \text{minimize} \quad & x_{j_1} + x_{j_2} \\ \text{subject to} \quad & a_{ij_1}x_{j_1} + a_{ij_2}x_{j_2} \geq r_i(j_1, j_2) \ \text{ for all } i \in M(j_1, j_2) \\ & x_{j_1}, x_{j_2} \geq 0, \end{aligned} \tag{2.15}$$

where several $O(n)$ algorithms have been developed for this problem [70][15]. This solution may yield overproduction δ_i of product $i \in M(j_1, j_2)$ as defined below.

$$\delta_i = a_{ij_1}x_{j_1} + a_{ij_2}x_{j_2} - r_i(j_1, j_2) \ \text{ for all } i \in M(j_1, j_2). \tag{2.16}$$

As overproduction is not desirable in order to minimize the number of stock rolls, we try to reduce the amount of overproduction. If a_{ij} is increased to $a_{ij} + 1$ for a bottleneck product i (i.e., $\delta_i = 0$ holds before modification), the resulting overproduction becomes $\delta_i = x_j$. From this observation, a simple rule (2.17) may be to select the cutting pattern $p_j \in \{p'_{j_1}, p'_{j_2}\}$ with smaller x_j and increase the a_{ij} of a bottleneck product i. Let i be the index of a bottleneck product, and assume $x_{j_1} \geq x_{j_2}$ without loss of generality.

$$a_{ij_2} := a_{ij_2} + 1. \tag{2.17}$$

But, this rule may keep selecting one particular cutting pattern in early stage of the algorithm, because the number of application x_j decreases further when a_{ij} is increased. To alleviate this drawback, we propose the following rule of perturbating a_{ij_1} and a_{ij_2}.

$$a_{ij_1} := a_{ij_1} + 1 \text{ and } a_{ij_2} := \max\left(0, a_{ij_2} - \left\lfloor \frac{x_{j_1}}{x_{j_2}} \right\rfloor\right). \tag{2.18}$$

The rule (2.17) or (2.18) can be applied if the cutting patterns p'_{j_1} and p'_{j_2} have enough residual lengths to permit the increase $a_{ij_1} := a_{ij_1} + 1$ or $a_{ij_2} := a_{ij_2} + 1$, respectively. If the rule (2.17) is applied, the resulting overproduction becomes $\delta_i = x_{j_2}$, while if the rule (2.18) is applied, we have $\delta_i = x_{j_1} - \min(a_{ij_2}x_{j_2}, \lfloor x_{j_1} \rfloor)$. If both rules are applicable, DAPG selects the one having smaller overproduction.

After modifying the cutting patterns by the above rule (2.17) or (2.18), and computing the numbers of their applications x_{j_1}, x_{j_2}, DAPG tries to readjust the residual lengths in the two cutting patterns, while satisfying all the demand constraints. The readjustment reduces the amount of overproduction and/or increases the residual lengths of the cutting patterns, where increasing residual lengths may be useful to eliminate other bottlenecks in subsequent steps. The readjustment is carried out in the following manner. Assume $x_{j_1} \geq x_{j_2}$ without loss of generality. Apply the following rules in the stated order if they are applicable (i.e., if cutting patterns have enough residual lengths in rule (2.21) and (2.22) below) and do not violate any demand constraint.

$$a_{ij_1} := a_{ij_1} - 1, \tag{2.19}$$

$$a_{ij_2} := a_{ij_2} - 1, \tag{2.20}$$

$$a_{ij_1} := a_{ij_1} - 1 \text{ and } a_{ij_2} := a_{ij_2} + 1, \tag{2.21}$$

$$a_{ij_1} := a_{ij_1} - 1 \text{ and } a_{ij_2} := a_{ij_2} - \left\lceil \frac{x_{j_1}}{x_{j_2}} \right\rceil. \tag{2.22}$$

The whole procedure is repeated as long as either the number of stock rolls $x_{ij_1} + x_{ij_2}$ decreases or the amount of overproduction:

$$\Delta = \sum_{i \in M(j_1, j_2)} \delta_i, \tag{2.23}$$

decreases, as a result of modification. When none of the two criteria are achieved, DAPG halts.

Now DAPG is described as follows, where L_1^{res} and L_2^{res} represent the residual lengths of p'_{j_1} and p'_{j_2}, respectively.

Algorithm DAPG

Input: Lengths l_i and demands d_i of all products $i \in M$, a set $M(j_1, j_2) \subseteq M$, residual demands $r_i(j_1, j_2)$ for all $i \in M(j_1, j_2)$, and the length of stock rolls L.

Output: A pair of cutting patterns $p'_{j_1} = (a_{ij_1} \mid i \in M(j_1, j_2))$ and $p'_{j_2} = (a_{ij_2} \mid i \in M(j_1, j_2))$.

Step 1: (Generation of a partition of $M(j_1, j_2)$)

Apply BALANCE-FIT to obtain a partition of $M(j_1, j_2)$ into a pair of disjoint subsets $M_1(j_1, j_2)$ and $M_2(j_1, j_2)$. If BALANCE-FIT outputs "failure", then output "failure" and halt.

Step 2: (Construction of an initial cutting patterns)

Construct a pair of cutting patterns p'_{j_1} and p'_{j_2} by applying (2.14), and compute $L_1^{res} := L - \sum_{i \in M_1(j_1,j_2)} a_{ij_1} l_i$, $L_2^{res} := L - \sum_{i \in M_2(j_1,j_2)} a_{ij_2} l_i$, $\delta_i := a_{ij_1} x_{j_1} + a_{ij_2} x_{j_2} - r_i(j_1, j_2)$ for all $i \in M(j_1, j_2)$, and $\Delta := \sum_{i \in M(j_1,j_2)} \delta_i$.

Step 3: (Elimination of bottleneck products)

Let $B \subseteq M(j_1, j_2)$ be the set of bottleneck products, i.e., $B := \{i \mid a_{ij_1} x_{j_1} + a_{ij_2} x_{j_2} = r_i(j_1, j_2), i \in M(j_1, j_2)\}$. For each $i \in B$, if $x_{j_2} < x_{j_1} - \min\left(x_{j_2}, \left\lfloor \frac{x_{j_1}}{x_{j_2}} \right\rfloor\right)$ and $l_i \leq L_2^{res}$ hold, apply rule (2.17) and let $L_2^{res} := L_2^{res} - l_i$. Otherwise if $l_i \leq L_1^{res}$ holds, apply rule (2.18), let $L_1^{res} := L_1^{res} - l_i$ and $L_2^{res} := L_2^{res} + \min\left(a_{ij_2}, \left\lfloor \frac{x_{j_1}}{x_{j_2}} \right\rfloor\right) l_i$.

Step 4: (Computing the numbers of applications x_{j_1}, x_{j_2})

Compute the real numbers of applications x_{j_1}, x_{j_2} to the cutting patterns p'_{j_1}, p'_{j_2} by solving the LP problem $\mathrm{LP}_2(j_1, j_2)$.

Step 5: (Readjustment of the cutting patterns)

For all $i \in M(j_1, j_2)$, apply (2.19)–(2.22) in this order if applicable without violating the length constraints of the cutting patterns and the demand constraints of all products.

Step 6: (Termination)

Compute the real numbers of applications x_{j_1}, x_{j_2} to the cutting patterns p'_{j_1}, p'_{j_2} by solving the LP problem $\mathrm{LP}_2(j_1, j_2)$. Compute the amount of overproduction Δ. If either the number of stock rolls $x_{j_1} + x_{j_2}$ or the amount of overproduction Δ decreases in the current loop of Steps 2–6, return to Step 2. Otherwise output the current pair of cutting patterns p'_{j_1}, p'_{j_2} together with the numbers of their applications x_{j_1}, x_{j_2}, and halt.

After completing DAPG, the cutting patterns may still have residual length enough to

accommodate some products $i \notin M(j_1, j_2)$. In this case, after computing the real numbers of applications $\overline{X}' = \{\overline{x}'_1, \overline{x}'_2, \ldots, \overline{x}'_n\}$ of the new set of cutting patterns $\Pi' = \Pi \cup \{p'_{j_1}, p'_{j_2}\} \setminus \{p_{j_1}, p_{j_2}\}$ by solving the LP relaxation LP(Π'), we try to reduce the number of stock rolls further by adding bottleneck products (i.e., product $i \in M$ such that $\sum_{p_j \in \Pi'} a_{ij}\overline{x}'_j = d_i$ holds) to such residual lengths (see Step 3 of LS-APG in the next section).

2.5 Entire Algorithm of Local Search

We now explain the framework of the local search algorithm LS-APG using MFF_INIT, SOLVE_IP and DAPG. Let (Π, X) be the current solution. LS-APG uses the first admissible move strategy by searching neighborhood $N_2^{apg}(\Pi)$ as follows. It first selects an index j_1, and then tries neighbor solutions $\Pi' = \Pi \cup \{p'_{j_1}, p'_{j_2}\} \setminus \{p_{j_1}, p_{j_2}\}$ for all $j_2 = j_1 + 1, j_1 + 2, \ldots, n, 1, \ldots, j_1 - 1$ (in this order). This is repeated for all $j_1 = 1, 2, \ldots, n$. If LS-APG finds a better solution (Π', X') in this process, it immediately replaces the current solution (Π, X) with the neighbor solution (Π', X'). To measure the improvement of solutions in LS-APG, we employ the main criterion of $f(\Pi, X)$ as well as the secondary criterion of $tloss(\Pi, X)$, defined by

$$tloss(\Pi, X) = \sum_{p_j \in \Pi} \left(L - \sum_{i \in M} a_{ij} l_i \right) x_j. \tag{2.24}$$

That is, LS-APG moves from (Π, X) to (Π', X') if either $f(\Pi, X) < f(\Pi', X')$ holds, or $f(\Pi, X) = f(\Pi', X')$ and $tloss(\Pi, X) < tloss(\Pi', X')$ hold.

Algorithm LS-APG

Input: Lengths l_i and demands d_i of all products $i \in M$, the number of different cutting patterns n, and the length of stock rolls L.

Output: A set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$, and the numbers of their applications $X = \{x_1, x_2, \ldots, x_n\}$, or "failure".

Step 1: (Generation of an initial solution)

Apply MFF_INIT to obtain an initial set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$. If MFF_INIT outputs "failure", then output "failure" and halt. Otherwise apply SOLVE_IP to compute the corresponding numbers of applications $X = \{x_1, x_2, \ldots, x_n\}$, set $j^* := 1$ and $j_1 := j^*$, and go to Step 2.

Step 2: (Construction of a neighbor solution Π')

Set $j_2 := j^* + 1 \pmod n$. Compute $r_i(j_1, j_2)$ of (2.11) for all $i \in M$, and let $M(j_1, j_2) := \{i \mid r_i > 0, i \in M\}$. Apply DAPG to obtain a pair of new cutting patterns p'_{j_1}, p'_{j_2}, and let $\Pi' = \Pi \cup \{p'_{j_1}, p'_{j_2}\} \setminus \{p_{j_1}, p_{j_2}\}$.

Step 3: (Elimination of bottleneck products)

Let $\overline{X}' = \{\overline{x}'_1, \overline{x}'_2, \ldots, \overline{x}'_n\}$ be the real numbers of applications to the new set of cutting patterns Π' obtained from the LP relaxation LP(Π'). Let B be the set of bottleneck products, i.e., $B := \{i \mid \sum_{p_j \in \Pi'} a_{ij}\overline{x}'_j = d_i\}$, and let $a_{kj} := a_{kj} + 1$ for all $k \in B$ for which there is a cutting pattern $p_j \in \Pi$ such that $l_k \leq L - \sum_{i \in M} a_{ij} l_i$. Let Π' be the resulting set of cutting patterns.

Step 4: (Move)

Compute the integer numbers of applications $X' = \{x'_1, x'_2, \ldots, x'_n\}$ to the new set of cutting patterns Π' by applying SOLVE_IP (see Section 2.3). If either $f(\Pi, X) < f(\Pi', X')$ holds, or $f(\Pi, X) = f(\Pi', X')$ and $tloss(\Pi, X) < tloss(\Pi', X')$ hold, then set $(\Pi, X) := (\Pi', X')$, $j^* := j^* + 1 \pmod n$, $j_1 := j^*$ and return to Step 2.

Step 5: (Termination)

Set $j_2 := j_2 + 1 \pmod n$. If $j_2 = j_1$ holds (i.e., all j_2 have been checked), set $j_1 := j_1 + 1 \pmod n$. If $j_1 = j^*$ holds (i.e., all j_1 and j_2 have been checked), output (Π, X) and halt. Otherwise return to Step 2.

2.6 Computational Experiment

We conducted computational experiment for random instances generated by CUTGEN [32], to compare LS-APG with the existing two algorithms SHP [46][47] and KOMBI [29].

SHP heuristically minimizes both the number of stock rolls f and the number of different cutting patterns n, and balances them by controlling the input parameter MAXTL (i.e., the upper bound of trim loss for new cutting patterns). KOMBI starts from a near optimal solution of the standard 1D-CSP (i.e., no restriction on the number of different cutting patterns) obtained by Stadtler's algorithm [83], and repeats reduction of the number of different cutting patterns by combining more than one cutting pattern into one. In other word, KOMBI tries to solve 1D-PMP that minimizes the number of different cutting patterns

while using a given number of stock rolls.

LS-APG and SHP were coded in C language and executed on an IBM-compatible personal computer (Pentium III 1GHz, 1GB memory). The results of KOMBI were taken from [29], as we could not get the source code of KOMBI. KOMBI was run on an IBM-compatible 486/66 personal computer using MODULA-2 as the programming language under MS-DOS 6.0.

We generated 18 classes of random instances by CUTGEN, which are defined by combining different values of parameters $L, m, \nu_1, \nu_2, \bar{d}$. The lengths l_i are treated as random variables taken from interval $[\nu_1 L, \nu_2 L]$. $\bar{d}$ is the average of demands $(d_1, d_2, \ldots, d_m)$ (the rule of generating d_i is described in [32]). In our computational experiments, L was set to 1000, m was set to 10, 20 and 40, and $\bar{d}$ was set to 10 and 100. Furthermore, (ν_1, ν_2) was set to (0.01, 0.2) for classes 1–6, (0.01, 0.8) for classes 7–12, and (0.2, 0.8) for classes 13–18. The parameter *seed* for generating random numbers was set to 1994. For each class, 100 problem instances were generated and tested. These classes of problem instances were also solved by KOMBI, where 100 instances are tested for each class.

As mentioned in Section 1.5, our local search algorithm can obtain a trade-off curve between the number of different cutting patterns n and the number of stock rolls f. For this, we conducted preliminary computational experiment. We took a random instance of class 12 generated by CUTGEN, and applied LS-APG for all n between n_{LB} and m, where

$$n_{LB} = \left\lceil \frac{\sum_{i \in M} l_i}{L} \right\rceil \tag{2.25}$$

is a trivial lower bound of different cutting patterns and m is the number of products. Figure 2.1 shows the number of stock rolls f with respect to the number of different cutting patterns n. As SHP has an input parameter MAXTL to control the maximum trim loss of cutting patterns, we also illustrated the solutions of SHP for different values of MAXTL. For this instance, we observe that LS-APG attains comparable number of stock rolls to SHP with smaller number of different cutting patterns. Figure 2.2 shows the CPU time of LS-APG for different n. We observe that the CPU time of LS-APG tends to increase as n increases.

Table 2.1 and 2.2 show computational results, where $\bar{n}_{LB}$ denotes the average of the lower bound n_{LB} on different cutting patterns, $\bar{n}$ denotes the average of different cutting patterns n, and $\bar{f}$ denotes the average of required stock rolls for each class. Table 2.1 contains the results of SHP and KOMBI, where SHP was run on three different values of MAXTL= 0.05, 0.03 and 0.01. The results of LS-APG are shown in Table 2.2. To capture the general behavior of trade-off curves of LS-APG, we show several points of trade-off curves that attain the

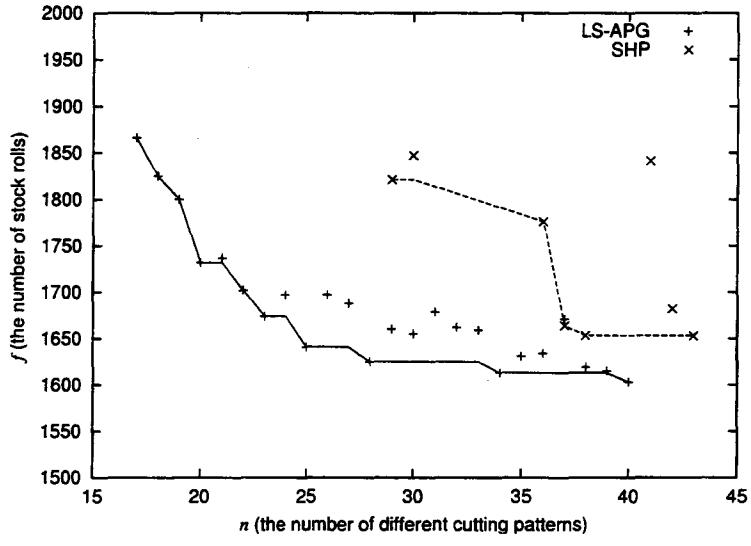

Figure 2.1: The number of stock rolls versus the number of different cutting patterns ($n_{LB} = 17$)

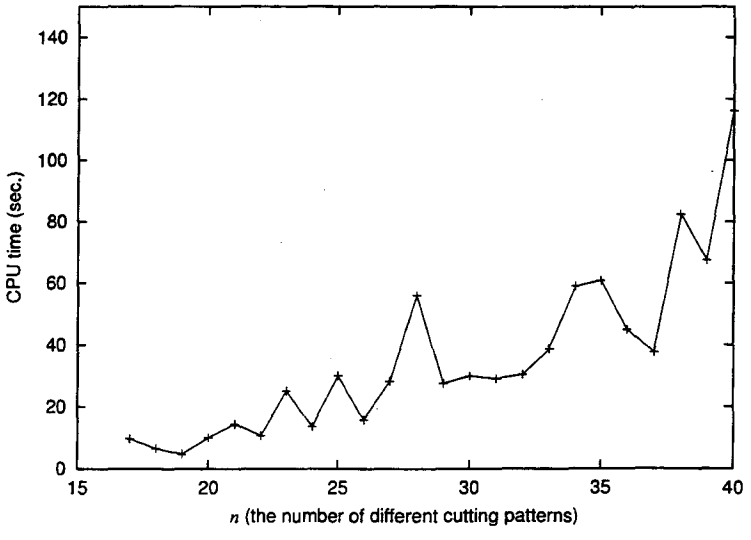

Figure 2.2: The CPU time in seconds versus the number of different cutting patterns

Table 2.1: Computational results of SHP and KOMBI for the random instances generated by CUTGEN

				SHP						KOMBI	
				MAXTL=0.05		0.03		0.01			
class	m	$\bar{d}$	$\bar{n}_{LB}$	$\bar{n}$	$\bar{f}$	$\bar{n}$	$\bar{f}$	$\bar{n}$	$\bar{f}$	$\bar{n}$	$\bar{f}$
1	10	10	1.67	4.08	11.68	4.25	11.62	4.49	11.57	3.40	11.49
2	10	100	1.67	6.33	112.80	6.33	111.81	6.75	110.85	7.81	110.25
3	20	10	2.56	5.77	22.55	5.89	22.37	5.98	22.17	5.89	22.13
4	20	100	2.56	9.06	220.63	8.98	218.94	9.25	217.00	14.26	215.93
5	40	10	4.26	9.07	43.89	9.03	43.60	9.01	43.17	10.75	42.96
6	40	100	4.26	13.90	434.59	13.45	430.79	13.77	426.81	25.44	424.71
7	10	10	4.62	10.14	52.21	10.33	52.19	10.82	52.77	7.90	50.21
8	10	100	4.62	11.30	519.88	11.46	520.36	11.97	526.81	9.96	499.52
9	20	10	8.65	18.58	97.42	19.22	97.96	19.91	98.82	15.03	93.67
10	20	100	8.65	20.96	970.43	21.74	973.63	21.99	984.32	19.28	932.32
11	40	10	16.27	35.06	186.45	36.29	187.37	37.78	189.91	28.74	176.97
12	40	100	16.27	39.90	1854.79	40.53	1865.41	41.86	1891.90	37.31	1766.20
13	10	10	5.54	10.55	65.46	10.55	65.41	10.67	65.52	8.97	63.27
14	10	100	5.54	10.92	652.95	10.95	654.80	11.12	654.95	10.32	632.12
15	20	10	10.52	20.11	123.36	20.30	124.36	20.86	124.60	16.88	119.93
16	20	100	10.52	21.09	1232.42	21.09	1240.39	21.32	1243.43	19.91	1191.80
17	40	10	19.85	38.02	235.64	38.52	238.06	39.10	239.77	31.46	224.68
18	40	100	19.85	40.16	2351.38	40.38	2366.95	40.67	2389.49	38.28	2242.40

Table 2.2: Computational results of LS-APG with different f_{UB} for the random instances generated by CUTGEN

				∞		$f^* + 0.05f_{LB}$		$f^* + 0.03f_{LB}$		$f^* + 0.01f_{LB}$	
class	m	$\bar{d}$	$\bar{n}_{LB}$	$\bar{n}$	$\bar{f}$	$\bar{n}$	$\bar{f}$	$\bar{n}$	$\bar{f}$	$\bar{n}$	$\bar{f}$
1	10	10	1.67	2.00	14.47	2.90	12.54	2.90	12.54	2.90	12.54
2	10	100	1.67	2.00	141.28	4.09	115.32	4.74	113.80	5.73	112.73
3	20	10	2.56	2.57	30.76	4.83	23.66	4.83	23.66	4.83	23.66
4	20	100	2.56	2.57	305.78	6.07	226.04	6.96	223.49	9.28	220.31
5	40	10	4.26	4.28	61.63	8.65	45.72	9.41	45.25	9.41	45.25
6	40	100	4.26	4.28	609.73	9.49	447.94	11.23	441.73	15.23	434.79
7	10	10	4.62	5.01	55.77	6.14	51.45	6.26	51.28	6.26	51.28
8	10	100	4.62	5.01	558.69	6.42	510.60	6.73	507.78	7.38	504.77
9	20	10	8.65	9.27	105.07	10.81	97.39	11.41	96.43	11.81	96.14
10	20	100	8.65	9.27	1053.08	11.44	962.27	12.22	953.92	13.91	944.45
11	40	10	16.27	16.95	201.46	19.60	185.68	20.77	183.79	23.98	181.34
12	40	100	16.27	16.95	2010.26	20.71	1837.90	22.44	1817.07	26.03	1787.39
13	10	10	5.54	6.26	68.73	7.06	64.45	7.30	64.36	7.30	64.36
14	10	100	5.54	6.26	687.14	7.19	644.87	7.45	641.05	7.77	638.53
15	20	10	10.52	11.76	129.10	12.76	123.43	13.33	122.65	14.18	121.81
16	20	100	10.52	11.76	1292.09	13.38	1226.05	14.01	1216.66	15.13	1205.10
17	40	10	19.85	21.50	246.34	22.89	235.85	23.49	233.55	25.58	230.94
18	40	100	19.85	21.50	2471.32	23.99	2334.88	25.35	2305.35	28.35	2274.79

minimum numbers of different cutting patterns n satisfying f_{UB} stock rolls or less. In this experiment, the upper bounds f_{UB} are set to as follows:

$$f_{UB} = f^* + \beta f_{LB}, \tag{2.26}$$

where f^* is the number of required stock rolls obtained by LS-APG for $n = m$, and f_{LB} is a trivial lower bound of required stock rolls:

$$f_{LB} = \left\lceil \frac{\sum_{i \in M} d_i l_i}{L} \right\rceil. \tag{2.27}$$

We tested LS-APG for $f_{UB} = \infty$ and $f_{UB} = f^* + \beta f_{LB}$ with $\beta = 0.05, 0.03, 0.01$.

From Table 2.1 and 2.2, we first observe that the numbers of stock rolls attained by KOMBI are smaller than those of SHP and LS-APG for all classes. However, SHP and LS-APG can obtain a wide variety of solutions by controlling their input parameters, i.e., they can realize trade-off curves between the number of different cutting patterns n and the number of stock rolls by their input parameters. SHP attains smaller number of different cutting patterns than KOMBI for classes 2–6 using only a slightly larger number of stock rolls, while the solutions of SHP are much worse than those of KOMBI for classes 7–18. It shows that SHP does not provide good trade-off curves for instances in which the ratio of product lengths l_i to the length of stock roll L is relatively large. On the other hand, LS-APG attains smaller number of different cutting patterns than KOMBI for all classes, without much increasing the number of additional stock rolls. Note that LS-APG obtains feasible solutions even for very small n, close to n_{LB}, while SHP and KOMBI could not produce feasible solutions for such n. From these observation, we may conclude that LS-APG is useful, as it can provide useful trade-off curves for a very wide range of n.

Table 2.6 gives the average CPU time of SHP (MAXTL=0.03), KOMBI and LS-APG ($f_{UB} = f^* + 0.03 f_{LB}$), respectively, for all classes. For these problem instances, SHP is faster than LS-APG except for classes 5 and 6, and KOMBI may be faster than LS-APG, taking account of the power of computers in use. However, the average CPU time of LS-APG is within 20 seconds for all classes, and it may be sufficiently short even if LS-APG repeatedly applied for all n in order to obtain trade-off curves.

Table 2.3: The average CPU time in seconds for the random instances generated by CUTGEN

class	m	$\bar{d}$	SHP	KOMBI	LS-APG
1	10	10	0.04	0.14	0.01
2	10	100	0.08	1.14	0.07
3	20	10	1.56	1.74	0.19
4	20	100	1.57	16.00	0.76
5	40	10	631.74	38.03	5.32
6	40	100	107.11	379.17	11.41
7	10	10	0.00	0.07	0.03
8	10	100	0.00	0.20	0.04
9	20	10	0.01	1.34	0.48
10	20	100	0.02	3.25	0.71
11	40	10	0.09	36.27	13.07
12	40	100	0.14	76.31	19.53
13	10	10	0.00	0.08	0.03
14	10	100	0.00	0.13	0.03
15	20	10	0.01	1.81	0.44
16	20	100	0.01	2.60	0.60
17	40	10	0.06	50.93	10.34
18	40	100	0.10	70.94	14.32

2.7 Conclusion

We proposed a local search algorithm based on an adaptive pattern generation (LS-APG) for 1D-PRP. It starts from an initial solution obtained by a modified first fit heuristic (MFF) known for the bin packing problem (BPP). Solutions in the neighborhood are defined by removing two cutting patterns from the current solution and adding two new cutting patterns from the set of candidate cutting patterns. However, the number of all feasible cutting patterns is too large to evaluate all of them, since it grows exponentially in the number of products. To facilitate the search in the neighborhood, we introduced the adaptive pattern generation (APG) to construct a small subset of the neighborhood containing good solutions. The adaptive pattern generation is based on the residual demands when two cutting patterns are removed from the current solution. We conducted computational experiment for random instances, and observed that LS-APG attains a wide variety of good solutions comparable to SHP and KOMBI, and LS-APG provides useful trade-off curves between the number of different cutting patterns and the number of required stock rolls for a very wide range.

Chapter 3

A Local Search Algorithm Based on Linear Programming Techniques

3.1 Introduction

n this chapter, we propose a local search algorithm based on linear programming techniques LS-LP). As mentioned in Section 1.5, we have to specify the following details to implement ocal search algorithms to 1D-PRP, i.e., (i) how to construct an initial solution, (ii) how to ompute auxiliary integer linear programming problems (ILP), and (iii) how to find promising olutions sufficiently among all neighbor solutions in the neighborhood. In Chapter 2, we roposed a heuristic algorithm to overcome them, called the adaptive pattern generation APG), and designed a local search algorithm based on the adaptive pattern generation (LS-APG). However, we could not attain sufficient efficiency since we still have a large number f LP relaxation problems to be solved in an execution of LS-APG. We try to overcome this difficulty by keeping an optimal simplex tableau in addition to the current solution; i.e., we tart the simplex algorithm from an optimal simplex tableau of the current solution, instead f starting it from scratch.

The proposed algorithm utilizes a dual optimal solution of LP relaxation as well as the olumn generation method proposed by Gilmore and Gomory [34][35], which is one of the ost representative linear programming approaches to the standard 1D-CSP (see Section

1.3). The column generation method utilizes a dual optimal solution of LP relaxation to determine a new cutting pattern to enter the basis of the LP relaxation; i.e., it gives us an aspiration that the new cutting pattern induces improvement of the current solution. It starts from a feasible solution of the standard 1D-CSP, and repeatedly adds new cutting patterns satisfying the aspiration of improvement by solving associated knapsack problems. As the column generation method never removes the cutting patterns in the current solution, it can always reduce the objective value $\bar{f}$ by adding new cutting patterns satisfying the aspiration of improvement. However, in 1D-PRP, in order to keep the number of different cutting patterns, we must remove a cutting pattern in the current solution while adding a new cutting pattern. Hence, we can not necessarily improve the objective value $\bar{f}$ by the new cutting pattern satisfying the aspiration of improvement. We propose a local search algorithm that uses the neighborhood obtained by adding a new cutting pattern satisfying the aspiration of improvement and removing a cutting pattern from the current solution; i.e., we test many pair of adding and removing cutting patterns efficiently by introducing the local search approach.

Finally, we conducted computational experiment for randomly generated problem instances, and observed that LS-LP attains a wide variety of solutions comparable to those of LS-APG, and it provides useful trade-off curves between the number of different cutting patterns and the number of required stock rolls.

3.2 Generation of an Initial Solution

As mentioned in Section 2.2, if there is no restriction on the number of different cutting patterns, as in the standard 1D-CSP, it is easy to construct a feasible solution. But just finding a feasible solution is not trivial in 1D-PRP, since it contains as a special case the bin packing problem (BPP). Hence, to design a local search algorithm to 1D-PRP, we first consider how to construct a feasible solution heuristically. Here, we assume that the number of different cutting patterns n is less than the number of products m, because otherwise a feasible solution is easily obtained. In this section, we develop two heuristic algorithms MFF (see Section 2.2) and UNIFORM-FIT to find a feasible solution (Π, X), where both algorithms are based on the first-fit algorithm (FF) known for BPP. Note that LS-LP directly uses the output of UNIFORM-FIT as an initial solution of the local search algorithm, while LS-APG improves the output of MFF by applying SAPG for each cutting pattern. We test local search

algorithms based on both algorithms for random instances in Section 3.7.

Let $\Pi = \{p_1, p_2, \ldots, p_n\}$ denote a set of cutting patterns, and $X = \{x_1, x_2, \ldots, x_n\}$ denote the numbers of their applications. Before starting UNIFORM-FIT, all cutting patterns $p_j \in \Pi$ are initialized to empty, i.e., $a_{ij} := 0$ for all $i \in M$. UNIFORM-FIT first sorts all products $i \in M$ in the descending order of demand d_i, and then assigns these products one by one in this order into the cutting pattern p_1 as long as the resulting sum of lengths does not exceed the length of stock roll L. In general steps for cutting patterns p_j ($j \geq 2$), UNIFORM-FIT sorts all products $i \in M$ in the ascending order of the number of their appearance in the set of cutting patterns $\{p_1, p_2, \ldots, p_{j-1}\}$ (while breaking ties in the descending order of demand d_i), and assigns them into the cutting pattern p_j in this order. Note that UNIFORM-FIT assigns at most once for each product $i \in M$ in one cutting pattern $p_j \in \Pi$, implying that the cutting patterns may have large trim losses since the minimization of the number of required stock rolls is not done by UNIFORM-FIT. Here, in the step of constructing p_j let L_j^{res} represent the residual length of the cutting pattern p_j, and λ_i represent the number of appearances of the product $i \in M$ in the set of cutting patterns $\{p_1, p_2, \ldots, p_{j-1}\}$:

$$\lambda_i = \sum_{p_k \in \{p_1, p_2, \ldots, p_{j-1}\}} a_{ik}. \tag{3.1}$$

Algorithm UNIFORM-FIT

Input: Lengths l_i and demands d_i of all products $i \in M$, the number of different cutting patterns n, and the length of stock roll L.

Output: A set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$, where $p_j = (a_{1j}, a_{2j}, \ldots, a_{mj})$.

Step 1: For all cutting patterns $p_j \in \Pi$, set $a_{ij} := 0$ for all $i \in M$ and $L_j^{res} := L$. Set $j := 1$.

Step 2: If $j = 1$ holds, sort all $i \in M$ in the descending order of demand d_i; otherwise compute the number of appearances λ_i for all $i \in M$, and sort them in the ascending order of λ_i, while breaking ties in the descending order of d_i. Let $\sigma(k)$ denote the k-th product in the resulting order. Set $k := 1$.

Step 3: If $l_{\sigma(k)} \leq L_j^{res}$ holds, set $a_{\sigma(k)j} := 1$ and $L_j^{res} := L_j^{res} - l_{\sigma(k)}$. If $k < m$ holds, set $k := k + 1$ and return to Step 3; otherwise go to Step 4.

Step 4: If $j = n$ holds, output $\Pi = \{p_1, p_2, \ldots, p_n\}$ and halt; otherwise set $j := j + 1$ and return to Step 2.

If there is a product $i \in M$ such that no cutting pattern $p_j \in \Pi$ contains it in the resulting set of cutting patterns Π, i.e., UNIFORM-FIT fails to obtain a feasible solution, we switch to the first-fit decreasing heuristic algorithm (FFD) known for BPP. FFD first sorts all products $i \in M$ in the descending order of l_i, and assigns them into the cutting pattern with the lowest index among those having the residual capacity of at least l_i. If this attempt also fails, we conclude "infeasibility" (i.e., failure).

Algorithm FFD

Input: Lengths l_i and demands d_i of all products $i \in M$, the number of different cutting patterns n, the length of stock rolls L.

Output: A set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$, where $p_j = (a_{1j}, a_{2j}, \ldots, a_{mj})$, or "failure".

Step 1: For all cutting patterns $p_j \in \Pi$, set $a_{ij} := 0$ for all $i \in M$, and $L_j^{res} := L$.

Step 2: Sort all products $i \in M$ in the descending order of l_i, where $\sigma(k)$ denotes the k-th product in this order. Set $k := 1$ and $j := 1$.

Step 3: If $l_{\sigma(k)} \leq L_j^{res}$ holds, set $a_{\sigma(k)j} := 1$, $L_j^{res} := L_j^{res} - l_{\sigma(k)}$, $k := k + 1$ and $j := 1$; otherwise set $j := j + 1$. If $k \leq m$ and $j \leq n$ hold, return to Step 3; otherwise go to Step 4.

Step 4: If $k > m$ holds, output $\Pi = \{p_1, p_2, \ldots, p_n\}$ and halt; otherwise output "failure" and halt.

Finally, we summarize the algorithm UNIFORM_INIT that constructs an initial feasible solution using UNIFORM-FIT and FFD.

Algorithm UNIFORM_INIT

Input: Lengths l_i and demands d_i of products $i \in M$, the number of different cutting patterns n, and the length of stock rolls L.

Output: A set of n cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$ and the numbers of their applications $X = \{x_1, x_2, \ldots, x_n\}$, or "failure".

Step 1: Apply UNIFORM-FIT to obtain a set of n cutting patterns. If this attempt fails, apply FFD. If FFD still outputs "failure", we output "failure" and halt.

Note that it is necessary to compute the numbers of applications x_j by SOLVE_IP (see Section 2.3).

3.3 Construction of the Neighborhood

A natural definition of neighborhood $N_1(\Pi)$ is given by replacing one cutting pattern $p_j \in \Pi$ with another cutting pattern $p'_j \in S \setminus \Pi$:

$$N_1(\Pi) = \{\Pi \cup \{p'_j\} \setminus \{p_j\} \mid p_j \in \Pi, p'_j \in S \setminus \Pi\}, \tag{3.2}$$

where S is the set of all feasible cutting patterns. As mentioned in Section 1.5, the number of all feasible cutting patterns $|S|$ exponentially grows as the number of products m, and most of them may not lead to improvement. To overcome this, we propose a new heuristic algorithm PERTURB, which is different from the adaptive pattern generation (APG) in Section 2.4. PERTURB generates a subset $N_1^{red}(\Pi)$ of the neighborhood $N_1(\Pi)$, containing good solutions. The construction is based on a dual optimal solution $\overline{Y} = \{\overline{y}_1, \overline{y}_2, \ldots, \overline{y}_m\}$ of the LP relaxation LP(Π). We now recall the LP relaxation LP(Π) of (2.8):

$$\begin{aligned} (\mathrm{LP}(\Pi)) \quad \text{minimize} \quad & \bar{f}(\Pi, X) = \sum_{j=1}^{n} x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j - s_i = d_i \ \text{ for } i = 1, \ldots, m \\ & x_j \geq 0 \ \text{ for } j = 1, 2, \ldots, n \\ & s_i \geq 0 \ \text{ for } i = 1, 2, \ldots, m, \end{aligned} \tag{3.3}$$

where s_i are the slack variables and represent the quantity of overproduction for products $i \in M$. Here, the dual problem DLP(Π) of the LP relaxation LP(Π) is described as follows:

$$\begin{aligned} (\mathrm{DLP}(\Pi)) \quad \text{maximize} \quad & \sum_{i=1}^{m} d_i y_i \\ \text{subject to} \quad & \sum_{i=1}^{m} a_{ij} y_i \leq 1 \ \text{ for } j = 1, 2, \ldots, n \\ & y_i \geq 0 \ \text{ for } i = 1, 2, \ldots, m. \end{aligned} \tag{3.4}$$

We consider the neighborhood $N_1^{red}(\Pi) \subseteq N_1(\Pi)$ as follows:

$$N_1^{red}(\Pi) = \{\Pi \cup \{p(i', j')\} \setminus \{p_{j'}\} \mid i' \in M'(\Pi),\ j' \in N\}, \tag{3.5}$$

where

$$M'(\Pi) = \{i \mid \overline{y}_i > 0,\ i \in M\}, \tag{3.6}$$

$$N = \{j \mid p_j \in \Pi\}, \tag{3.7}$$

and $p(i', j') = (a'_{ij'} \mid i \in M)$ is the new cutting pattern generated by PERTURB for a pair of i' and j'. The dual variables y_i give us information of how much improvement can be induced by slightly reducing the right hand side d_i of the i-th constraint of LP(Π). The reduction of d_i may be also achieved by increasing an a_{ij} in the i-th constraint of LP(Π). In this sense, y_i may serve as an indication of the effectiveness of increasing a_{ij} in a cutting pattern $p_j \in \Pi$. Note that the number of positive $\overline{y}_i$ of an optimal solution is at most the number of cutting patterns n, and hence the size of $N_1^{red}(\Pi)$ is $O(n^2)$.

Based on this observation, we now explain the algorithm PERTURB. It is executed for a pair of i' and j' to generate a new cutting pattern $p(i', j')$, which is obtained by increasing $a_{i'j'}$ and decreasing some $a_{ij'} (i \neq i')$ with smaller $\overline{y}_i / l_i$ (to recover feasibility of the cutting pattern (1.3)). For this, PERTURB first sorts all products $i \in M$ in the ascending order of $\overline{y}_i / l_i$ while using the descending order of the overproduction s_i as the secondary criterion for $i \in M \setminus M'(\Pi)$ (i.e., $\overline{y}_i = 0$ holds). Then, PERTURB adds one product $i' \in M'(\Pi)$ to the cutting pattern $p_{j'} \in \Pi$ (i.e., $a'_{i'j'} := a_{i'j'} + 1$). If the new cutting pattern $p(i', j')$ violates the constraint (1.3), PERTURB sequentially removes other products in the above order until the cutting pattern $p(i', j')$ satisfies (1.3), and then adds back the products in the reversed order as long as the cutting pattern $p(i', j')$ still satisfies (1.3). Here, let L^{res} represent the residual length of the cutting pattern $p(i', j')$.

Algorithm PERTURB

Input: A product $i' \in M$, a cutting pattern $p_{j'} = (a_{1j'}, a_{2j'}, \ldots, a_{mj'}) \in \Pi$, lengths l_i and demands d_i of all products $i \in M$, the length of stock rolls L, and a dual optimal solution $\overline{Y} = \{\overline{y}_1, \overline{y}_2, \ldots, \overline{y}_m\}$ of DLP(Π).

Output: A new cutting pattern $p(i', j') = (a'_{1j'}, a'_{2j'}, \ldots, a'_{mj'})$ or "failure".

Step 1: Sort all $i \in M$ in the ascending order of $\overline{y}_i / l_i$ while using the descending order of overproduction s_i as the secondary criterion for $i \in M \setminus M'(\Pi)$. Let $\sigma(k)$ denote the k-th product in the resulting order.

Step 2: Set $a'_{i'j'} := a_{i'j'} + 1$, $a'_{i'j'} := a_{i'j'}$ for all $i \in M \setminus \{i'\}$, and $L^{res} := L - \sum_{i \in M} a'_{ij'} l_i$. Set $k := 1$.

Step 3: If $L^{res} \geq 0$ holds, set $k := m$ and go to Step 5; otherwise if $k > m$ holds, output "failure" and halt.

Step 4: If $\sigma(k) \neq i'$ and $a'_{\sigma(k)j'} > 0$ hold, set $a'_{\sigma(k)j'} := a'_{\sigma(k)j'} - 1$, and $L^{res} := L^{res} + l_{\sigma(k)}$. Set $k := k + 1$ and return to Step 3.

Step 5: If $l_{\sigma(k)} \leq L^{res}$ holds, set $a'_{\sigma(k)j'} := a'_{\sigma(k)j'} + 1$ and $L^{res} := L^{res} - l_{\sigma(k)}$. If $k > 1$ holds, set $k := k - 1$ and return to Step 5; otherwise output the resulting the cutting pattern $p(i', j') = (a'_{1j'}, a'_{2j'}, \ldots, a'_{mj'})$ and halt.

Here, we note that the set $M'(\Pi)$ is also given by

$$M'(\Pi) = \{i \mid s_i = 0,\ i \in M\}, \tag{3.8}$$

which is obtained by the complementary slackness of linear programming. In this sense, PERTURB tends to eliminate the bottleneck products (i.e., $s_i = 0$ holds) as well as the adaptive pattern generation of LS-APG (see Section 2.4) does.

Now we recall the column generation method to the standard 1D-CSP in Section 1.3. The column generation method also utilizes an optimal solution $\overline{Y} = \{\overline{y}_1, \overline{y}_2, \ldots, \overline{y}_m\}$ of DLP(Π) to generate a new cutting pattern. It uses the following aspiration to induce improvement of the current solution:

$$z(\Pi, p') = \sum_{i \in M} \overline{y}_i a'_i, \tag{3.9}$$

where $p' = \{a'_1, a'_2, \ldots, a'_m\}$ represents the new cutting pattern. If the new cutting pattern p' satisfies $z(\Pi, p') > 1$, we can improve the current solution by adding the new cutting pattern. However, in 1D-PRP, in order to keep the number of different cutting patterns, we must remove a cutting pattern in the current solution while adding a new cutting pattern, i.e., PERTURB substantially replaces the cutting pattern $p_{j'}$ in the current solution with the new cutting pattern $p(i', j')$. Figure 3.3 shows the change of the dual optimal solution by applying PERTURB. It is observed that the objective value $\bar{f}$ may increase even if the new cutting pattern $p(i', j')$ satisfies $z(\Pi, p(i', j')) > 1$.

On the other hand, if the new cutting pattern $p(i', j')$ satisfies $z(\Pi, p(i', j')) \leq 1$, it never lead to improvement of the current solution. In this case, we prepare another pattern generation algorithm PATGEN which generates a new cutting pattern $p(i', j')$ from scratch.

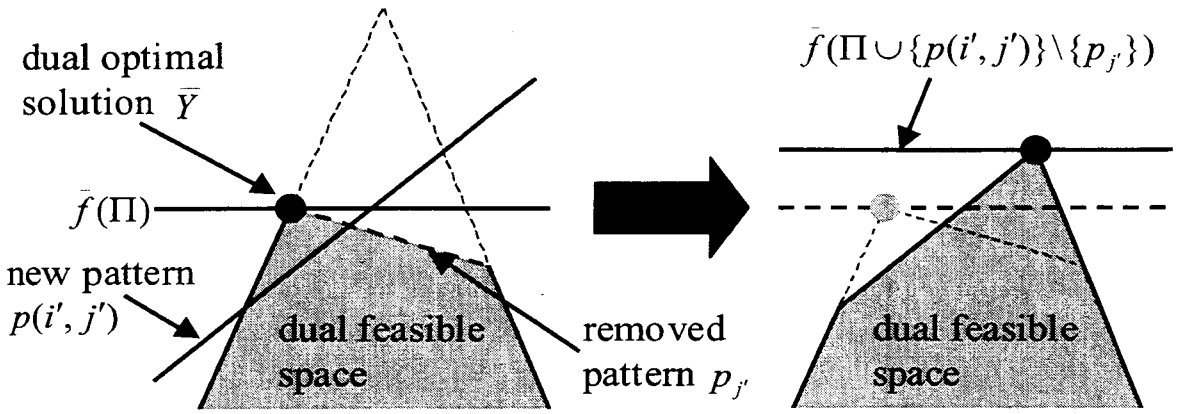

Figure 3.1: The change of the dual optimal solution by PERTURB

PATGEN first sorts all products $i \in M'(\Pi)$ in the descending order of $\overline{y}_i/l_i$, and repeatedly adds them in this order until $\sum_{i \in M} a'_{ij'}\overline{y}_i > 1$ holds.

Algorithm PATGEN

Input: A cutting pattern $p_{j'} = (a_{1j'}, a_{2j'}, \ldots, a_{mj'}) \in \Pi$, lengths l_i and demands d_i of all products $i \in M$, the length of stock rolls L, and a dual optimal solution $\overline{Y} = \{\overline{y}_1, \overline{y}_2, \ldots, \overline{y}_m\}$ of DLP(Π).

Output: A new cutting pattern $p(i', j') = (a'_{1j'}, a'_{2j'}, \ldots, a'_{mj'})$ or "failure".

Step 1: Sort all $i \in M'(\Pi)$ in the descending order of $\overline{y}_i/l_i$. Let $\sigma(k)$ denote the k-th product in the resulting order.

Step 2: Set $a'_{ij'} := 0$ for all $i \in M$, $L^{res} := L$, and $k := 1$.

Step 3: If $L^{res} \leq \min_{i \in M'(\Pi)} l_i$ or $\sum_{i \in M} a'_{ij'}\overline{y}_i > 1$ holds, output the resulting cutting pattern $p(i', j') = (a'_{1j'}, a'_{2j'}, \ldots, a'_{mj'})$ and halt; otherwise if $l_{\sigma(k)} \leq L^{res}$ holds, set $a'_{\sigma(k)j'} := a'_{\sigma(k)j'} + 1$ and $L^{res} := L^{res} - l_{\sigma(k)}$. If $k = n$ holds, set $k := 1$; otherwise $k := k + 1$. Return to Step 3.

3.4 Solving many LP Relaxations

For a given set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$, the problem of computing the numbers of their applications $X = \{x_1, x_2, \ldots, x_n\}$ to minimize the cost function $f(\Pi, X)$

can be formulated as the following integer linear programming problem (ILP):

$$\begin{aligned} (\text{ILP}(\Pi)) \quad \text{minimize} \quad & f(\Pi, X) = \sum_{j=1}^{n} x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j - s_i = d_i \quad \text{for } i = 1, \ldots, m \\ & x_j \in \boldsymbol{Z}_+ \quad \text{for } j = 1, 2, \ldots, n \\ & s_i \in \boldsymbol{Z}_+ \quad \text{for } i = 1, 2, \ldots, m, \end{aligned} \tag{3.10}$$

As mentioned in Section 2.3, this problem contains the set covering problem (SCP) known to be NP-hard [31] as a special case. Hence, we propose a heuristic algorithm SOLVE_IP to find an approximate solution $\widehat{X} = \{\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n\}$, which is based on an optimal solution of its LP relaxation $\overline{X} = \{\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\}$. Indeed we can compute the number of applications $\widehat{X}$ for a set of cutting patterns Π in relatively small CPU time. However, in order to determine one move operation, we still have a large number of LP relaxations LP(Π) to be solved. For example, LS-LP needs to solve $O(n^2)$ LP relaxations, and it is quite time consuming. In this section, we consider an exact algorithm SOLVE_LP to solve LP relaxation LP(Π).

Let $\Pi' = \Pi \cup \{p(i', j')\} \setminus \{p_{j'}\}$ denote a solution in the neighborhood $N_1^{red}(\Pi)$, where $p(i', j')$ is the new cutting pattern generated by PERTURB. Since the instance LP(Π') is only slightly different from LP(Π), we can start simplex algorithm from the optimal tableau of the instance LP(Π), instead of starting it from scratch. Now we consider an optimal solution $\overline{X} = \{\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\}$ of LP(Π) and the optimal values of its slack variables $\{\overline{s}_1, \overline{s}_2, \ldots, \overline{s}_m\}$. Here, we use $\overline{x}_j > 0$ and $\overline{s}_k > 0$ (resp., $\overline{x}_j = 0$ and $\overline{s}_k = 0$) to denote basic (resp., non-basic) variables. The cutting patterns in the current solution $p_j = (a_{j1}, a_{2j}, \ldots, a_{mj})$ correspond to the columns of the simplex tableau of LP(Π). Let w_k and $q_k = (b_{k1}, b_{2k}, \ldots, b_{mk})$ denote the coefficient of the cost function and the columns corresponding to the slack variables $\overline{s}_k$. Let $\boldsymbol{c}_B$ and $\boldsymbol{B}$ (resp., $\boldsymbol{c}_N$ and $\boldsymbol{N}$) denote the coefficients of the cost function and the columns corresponding to the basic (resp., non-basic) variables, i.e., $\boldsymbol{c}_B = (c_j \mid \overline{x}_j > 0) \cup (w_k \mid \overline{s}_k > 0)$, $\boldsymbol{B} = (p_j^t \mid \overline{x}_j > 0) \cup (q_k^t \mid \overline{s}_k > 0)$, $\boldsymbol{c}_N = (c_j \mid \overline{x}_j = 0) \cup (w_k \mid \overline{s}_k = 0)$, and $\boldsymbol{N} = (p_j^t \mid \overline{x}_j = 0) \cup (q_k^t \mid \overline{s}_k = 0)$.

The optimal simplex tableau $T = (\tilde{\boldsymbol{B}}, \tilde{\boldsymbol{N}}, \tilde{\boldsymbol{c}}_B, \tilde{\boldsymbol{c}}_N)$ contains the basic columns $\tilde{\boldsymbol{B}} = (\tilde{p}_j \mid \overline{x}_j > 0) \cup (\tilde{q}_k \mid \overline{s}_k > 0)$, the non-basic columns $\tilde{\boldsymbol{N}} = (\tilde{p}_j \mid \overline{x}_j = 0) \cup (\tilde{q}_k \mid \overline{s}_k = 0)$, the reduced costs corresponding to the basic variables $\tilde{\boldsymbol{c}}_B = (\tilde{c}_j \mid \overline{x}_j > 0) \cup (\tilde{w}_k \mid \overline{s}_k > 0)$, and the reduced costs corresponding to the non-basic variables $\tilde{\boldsymbol{c}}_N = (\tilde{c}_j \mid \overline{x}_j = 0) \cup (\tilde{w}_k \mid \overline{s}_k = 0)$.

where the columns $\tilde{p}_j = (\tilde{a}_{1j}, \tilde{a}_{2j}, \ldots, \tilde{a}_{mj})^t$ correspond to the variables x_j and the columns $\tilde{q}_k = (\tilde{b}_{1k}, \tilde{b}_{2k}, \ldots, \tilde{b}_{mk})^t$ correspond to the variables s_k. They are computed by

$$\tilde{p}_j = \boldsymbol{B}^{-1} p_j^t \tag{3.11}$$

$$\tilde{q}_k = \boldsymbol{B}^{-1} q_k^t \tag{3.12}$$

$$\tilde{c}_j = 1 - \sum_{i \in M} \overline{y}_i a_{ij} \tag{3.13}$$

$$\tilde{w}_k = -\sum_{i \in M} \overline{y}_i b_{ik}, \tag{3.14}$$

where $\overline{Y} = \{\overline{y}_1, \overline{y}_2, \ldots, \overline{y}_m\}$ is an optimal solution of DLP(Π) and it can be easily computed based on the duality theorem:

$$\overline{Y} = \boldsymbol{c}_B^t \boldsymbol{B}^{-1}. \tag{3.15}$$

To construct a new simplex tableau $T' = (\tilde{\boldsymbol{B}}', \tilde{\boldsymbol{N}}', \tilde{\boldsymbol{c}}_B', \tilde{\boldsymbol{c}}_N')$ of the new set of cutting patterns Π', we first add the new column $\tilde{p}(i', j')$ corresponding to the new cutting pattern $p(i', j') = (a'_{1j'}, a'_{2j'}, \ldots, a'_{mj'})$ generated by PERTURB:

$$\tilde{p}(i', j') = \boldsymbol{B}^{-1} p(i', j')^t. \tag{3.16}$$

The reduced cost $\tilde{c}'_{j'}$ and the number of applications corresponding to the new cutting pattern $p(i', j')$ are also computed as follows:

$$\tilde{c}'_{j'} = 1 - \sum_{i \in M} \overline{y}_i a'_{ij'} \tag{3.17}$$

$$\overline{x}'_{j'} = 0. \tag{3.18}$$

We remove the column $\tilde{p}_{j'}$ from basis and add the new column $\tilde{p}(i', j')$ to basis by applying a pivoting operation to exchange $\overline{x}_{j'}$ and $\overline{x}'_{j'}$ (consequently $\overline{x}_{j'}$ becomes 0 and $\overline{x}'_{j'}$ becomes nonnegative). Figure 3.4 represents the above operations to the optimal simplex tableau T. Then we apply the criss-cross algorithm [103] to the resulting simplex tableau. Even if this starting simplex tableau is neither feasible (i.e., there is at least one $\overline{x}_j < 0$, $\overline{s}_k < 0$ or $\overline{x}'_{j'} < 0$) nor dual feasible (i.e., there is at least one $\tilde{c}_j < 0$, $\tilde{w}_k < 0$ or $\tilde{c}'_{j'} < 0$), the criss-cross algorithm is guaranteed to converge to an optimal solution of LP(Π') often in a small number of pivoting operations.

Algorithm SOLVE_LP

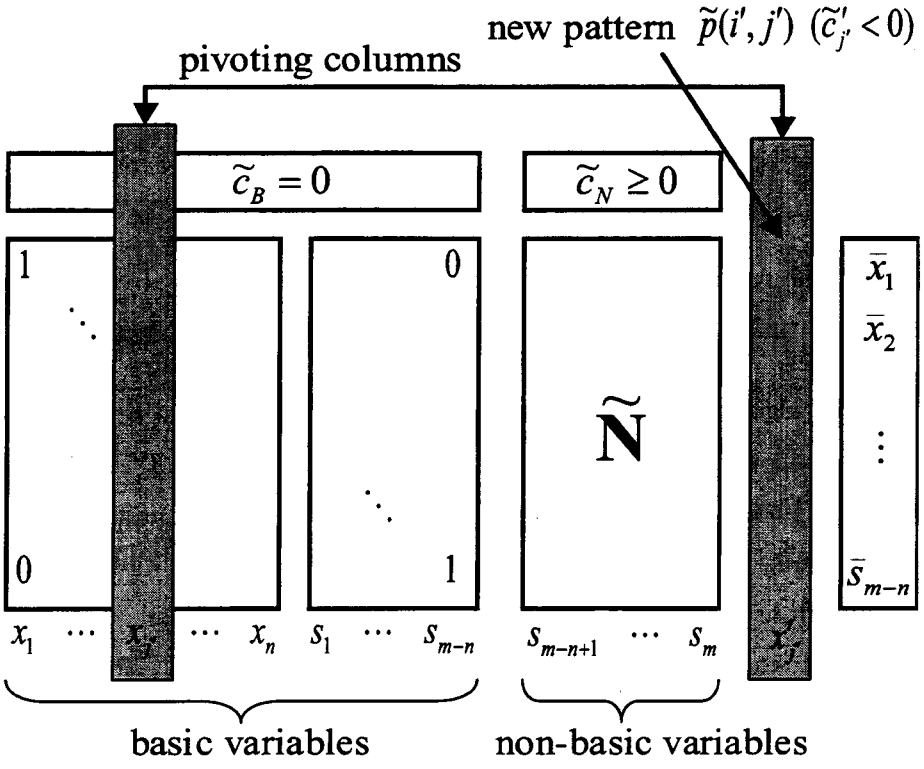

Figure 3.2: Exchanging the columns $\tilde{p}_{j'}$ and $\tilde{p}(i',j')$ in the optimal simplex tableau T

Input: An optimal solution $\overline{X} = \{\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_{j'}, \overline{x}_n\}$ together with its simplex tableau $T = (\tilde{\boldsymbol{B}}, \tilde{\boldsymbol{N}}, \tilde{\boldsymbol{c}}_B, \tilde{\boldsymbol{c}}_N)$ of LP(Π), where $\Pi = \{p_1, p_2, \ldots, p_n\}$ is the current set of patterns, and a new pattern $p(i', j') = (a'_{1j'}, a'_{2j'}, \ldots, a'_{mj'})$.

Output: An optimal solution $\overline{X}' = \{\overline{x}'_1, \overline{x}'_2, \ldots, \overline{x}'_{j'}, \ldots, \overline{x}'_n\}$ and its simplex tableau $T' = (\tilde{\boldsymbol{B}}', \tilde{\boldsymbol{N}}', \tilde{\boldsymbol{c}}'_B, \tilde{\boldsymbol{c}}'_N)$ of LP(Π') for the new set of patterns $\Pi' = \Pi \cup \{p(i', j')\} \setminus \{p_{j'}\}$, or "failure".

Step 1: Add $\tilde{p}(i', j') := \boldsymbol{B}^{-1} p(i', j')^t$, $\overline{x}'_{j'} := 0$, $\tilde{c}'_{j'} := 1 - \sum_{i \in M} \overline{y}_i a'_{ij'}$ to the simplex tableau T.

Step 2: Apply the pivoting operation to exchange columns $\tilde{p}_{j'}$ and $\tilde{p}(i', j')$. If the resulting simplex tableau is feasible, apply the revised simplex algorithm; otherwise apply the criss-cross algorithm to obtain an optimal solution $\overline{X}'$ and its tableau T'.

Step 3: If the resulting solution $\overline{X}'$ contains at least one negative variable $\overline{x}_j$, outputs "failure"; otherwise output $\overline{X}'$ and its simplex tableau T'. Halt.

The criss-cross algorithm does not monotonously decrease the objective value $\tilde{f}(\Pi, X)$. It shows that we can not necessarily obtain an improving solution (Π', X') of LP(Π) (i.e., $\tilde{f}(\Pi', X') < \tilde{f}(\Pi, X)$ holds) by applying a new cutting pattern $p(i', j')$ satisfying $z(\Pi, p(i', j')) > 1$. On the other hand, if the resulting simplex tableau is feasible (i.e., all basic variables take nonnegative values), as the revised simplex algorithm decreases monotonously the objective value $\tilde{f}(\Pi, X)$, and we can always obtain an improving solution (Π', X') of LP(Π). From this observation, we see that the feasibility of the resulting simplex tableau is a sufficient condition to obtain an improving solution. If the resulting simplex tableau is feasible, the revised simplex algorithm is usually faster than the criss-cross algorithm. Therefore, in such case, we use the revised simplex algorithm instead of the criss-cross algorithm.

To see the effectiveness of SOLVE_LP, we compared two types of LS-LP, one is entirely based on the revised simplex algorithm (i.e., starting from scratch and the other is based on SOLVE_LP (i.e., starting from the optimal simplex tableau of the previous solution). We tested two algorithms on 18 classes of randomly generated instances (see Section 2.6). Table 3.1 shows the number of pivoting operations required to solve one LP relaxation, averaged over 10 instances for each class, where n (i.e., the number of different cutting patterns) is set to those of SHP solutions.

Table 3.1: The average number of pivoting operations for solving single LP(Π)

class	m	$\bar{d}$	revised simplex	SOLVE_LP
1	10	10	11.86	2.83
2	10	100	14.32	2.90
3	20	10	26.07	3.81
4	20	100	31.91	5.61
5	40	10	52.47	3.95
6	40	100	82.49	10.21
7	10	10	11.99	1.27
8	10	100	14.14	0.88
9	20	10	19.46	0.88
10	20	100	31.48	1.19
11	40	10	52.16	0.84
12	40	100	73.23	1.64
13	10	10	11.49	1.06
14	10	100	12.64	1.30
15	20	10	22.53	0.79
16	20	100	23.14	1.01
17	40	10	42.23	0.60
18	40	100	44.96	1.27

From Table 3.1, the number of pivoting operations of the revised simplex algorithm is roughly proportional to the number of product m, but that of SOLVE_LP is almost constant and is much smaller than that of the revised simplex algorithm. It is also observed that SOLVE_LP often attains an optimal solution without applying any pivoting operation. Since local search algorithms to 1D-PRP consumes most of CPU time for solving LP relaxations, SOLVE_LP is very important to make them practical.

3.5 Entire Algorithm of Local Search

We now explain the framework of the local search algorithm LS-LP using UNIFORM_INIT (or MFF_INIT), SOLVE_IP (based on SOLVE_LP), PERTURB and PATGEN as its components. Let (Π, X) be the current solution. It adopts the first admissible move strategy, i.e., if LS-LP finds a better solution (Π', X') in its neighborhood $N_1^{red}(\Pi)$, LS-LP immediately moves to it, and repeats the neighborhood search from there. To measure improvement of solutions in LS-LP, we employ the main criterion of $f(\Pi, X)$ as well as the secondary criterion of the total overproduction:

$$\Delta(\Pi, X) = \sum_{i \in M} \delta_i \tag{3.19}$$

$$\delta_i = \max\left\{0, \sum_{p_j \in \Pi} a_{ij} x_j - d_i\right\}. \tag{3.20}$$

That is, LS-LP moves from (Π, X) to (Π', X') if either $f(\Pi', X') < f(\Pi, X)$ holds, or $f(\Pi', X') = f(\Pi, X)$ and $\Delta(\Pi', X') < \Delta(\Pi, X)$ hold.

Algorithm LS-LP

Input: Lengths l_i and demands d_i of all products $i \in M$, the number of different cutting patterns n, and the length of stock rolls L.

Output: A set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$ and the numbers of their applications $X = \{x_1, x_2, \ldots, x_n\}$, or "failure".

Step 1: Apply UNIFORM_INIT or MFF_INIT to obtain an initial set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$, and apply SOLVE_IP to compute the numbers of their applications $X = \{x_1, x_2, \ldots, x_n\}$. If SOLVE_IP outputs "failure", output "failure" and halt; otherwise set $i^* := 1$, $j^* := 1$, $i' := i^*$ and $j' := j^*$.

Step 2: If $i' \in M'(\Pi)$ holds, apply PERTURB to generate the new cutting pattern $p(i', j') = (a'_{1j'}, a'_{2j'}, \ldots, a'_{mj'})$; otherwise go to Step 5. If $\sum_{i \in M} a'_{ij'} \overline{y}_i \leq 1$ holds for the new cutting pattern $p(i', j')$, apply PATGEN to reconstruct the new cutting pattern $p(i', j')$. Let $\Pi' := \Pi \cup \{p(i', j')\} \setminus \{p_{j'}\}$.

Step 3: For the set of cutting patterns $\Pi' = \{p'_1, p'_2, \ldots, p'_n\}$, compute the numbers of their applications $X' = \{x'_1, x'_2, \ldots, x'_n\}$ by SOLVE_IP (which employs SOLVE_LP). If SOLVE_IP outputs "failure", go to Step 5.

Step 4: If either $f(\Pi', X') < f(\Pi, X)$ holds, or $f(\Pi', X') = f(\Pi, X)$ and $\Delta(\Pi', X') < \Delta(\Pi, X)$ hold, move to the new solution; i.e., set $(\Pi, X) := (\Pi', X')$. Set $i^* := i^* + 1 \pmod m$, $j^* := j^* + 1 \pmod n$, $i' := i^*$, $j' := j^*$, and return to Step 2.

Step 5: Set $i' := i' + 1 \pmod m$. If $i' = i^*$ holds (i.e., all i' have been checked), set $j' := j' + 1 \pmod n$. If $j' = j^*$ holds (i.e., all i' and j' have been checked), output (Π, X) and halts; otherwise return to Step 2.

3.6 Iterated Local Search Algorithm

Since the neighborhood $N_1^{red}(\Pi)$ is rather small, LS-LP often converges to a poor local optimal solution after a small number of move operations. To overcome such phenomenon, we introduce an extension of the local search algorithm called the iterated local search algorithm (ILS) (see Section 1.2). We now consider the iterated version of LS-LP (ILS-LP) in this section. ILS-LP first applies the simple local search algorithm LS-LP to an initial solution generated by UNIFORM_INIT (or MFF_INIT). It then repeats LS-LP from different initial solutions generated by perturbating the best solution obtained by then. The perturbation is done by a random move to a neighbor solution $\Pi' \in N_2^{swap}(\Pi)$, where $N_2^{swap}(\Pi)$ is another type of neighborhood defined as follows:

$$N_2^{swap}(\Pi) = \{\Pi \cup \{q(i', j'_1), q(i', j'_2)\} \setminus \{p_{j'_1}, p_{j'_2}\} \mid i' \in M,\ j'_1, j'_2 \in N\}, \tag{3.21}$$

where

$$N = \{j \mid p_j \in \Pi\}, \tag{3.22}$$

and $q(i', j'_1)$ and $q(i', j'_2)$ are the new cutting patterns generated by a heuristic algorithm SWAP for the given $i' \in M$ and $j'_1, j'_2 \in N$.

Given j'_1 and j'_2, assume without loss of generality that the cutting patterns $p_{j'_1}$ and $p_{j'_2}$ satisfy $x_{j'_1} \geq x_{j'_2}$. If the overproduction of the product i' takes positive value (i.e., $\delta_{i'} > 0$), SWAP increases the product i' in the cutting pattern $p_{j'_2}$ (i.e., $a_{i'j'_2} := a_{i'j'_2} + 1$), and decreases the product i' in the cutting pattern $p_{j'_1}$ (i.e., $a_{i'j'_1} := a_{i'j'_1} - 1$), if $a_{i'j'_1} > 0$ holds. If the cutting pattern $p_{j'_2}$ violates the pattern feasibility (1.3), SWAP moves some products $i \in M \setminus \{i'\}$ from $p_{j'_2}$ to $p_{j'_1}$ in the ascending order of overproduction δ_i until the cutting pattern $p_{j'_2}$ satisfies (1.3) (in this case, if the cutting pattern $p_{j'_1}$ has no residual length to accommodate the products from $p_{j'_2}$, SWAP only removes the products from $p_{j'_2}$). Here, $L^{res}_{j'_1}$ and $L^{res}_{j'_2}$ represent the residual lengths of the cutting patterns $q(i', j'_1)$ and $q(i', j'_2)$, respectively.

Algorithm SWAP

Input: A product $i' \in M$ satisfying $\delta_{i'} > 0$, a pair of cutting patterns $p_{j'_1} = (a_{1j'_1}, a_{2j'_1}, \ldots, a_{mj'_1})$ and $p_{j'_2} = (a_{1j'_2}, a_{2j'_2}, \ldots, a_{mj'_2})$, the number of their applications $x_{j'_1}$ and $x_{j'_2}$, where $x_{j'_1} \geq x_{j'_2}$. Lengths l_i and overproduction δ_i of all products $i \in M$ and the length of stock rolls L.

Output: A pair of cutting patterns $q(i', j'_1) = (a'_{1j'_1}, a'_{2j'_1}, \ldots, a'_{mj'_1})$ and $q(i', j'_2) = (a'_{1j'_2}, a'_{2j'_2}, \ldots, a'_{mj'_2})$, or "failure".

Step 1: Sort all $i \in M \setminus \{i'\}$ in the ascending order of overproduction δ_i. Let $\sigma(k)$ denote the k-th product in the resulting order. Set $L^{res}_{j'_1} := L - \sum_{i \in M} a_{ij'_1} l_i$ and $L^{res}_{j'_2} := L - \sum_{i \in M} a_{ij'_2} l_i$.

Step 2: If $a_{i'j'_1} = 0$ holds, output "failure" and halt; otherwise set $a'_{i'j'_1} := a_{i'j'_1} - 1$, $a'_{ij'_1} := a_{ij'_1}$ for all $i \in M \setminus \{i'\}$, and $L^{res}_{j'_1} := L^{res}_{j'_1} + l_{i'}$. Set $a'_{i'j'_2} := a_{i'j'_2} + 1$, $a'_{ij'_2} := a_{ij'_2}$ for all $i \in M \setminus \{i'\}$, and $L^{res}_{j'_2} := L^{res}_{j'_2} - l_{i'}$. Set $k := 1$.

Step 3: If $L^{res}_{j'_2} \geq 0$ holds, output $q(i', j'_1)$ and $q(i', j'_2)$, and halt; else if $k > m$ holds, output "failure" and halt. Otherwise go to Step 4.

Step 4: If $\sigma(k) \neq i'$ and $a_{\sigma(k)j'_2} > 0$ hold, set $a'_{\sigma(k)j'_2} := a_{\sigma(k)j'_2} - 1$, $L^{res}_{j'_2} := L^{res}_{j'_2} + l_{\sigma(k)}$ and if $l_{\sigma(k)} \leq L^{res}_{j'_1}$ holds, set $a'_{\sigma(k)j'_1} := a_{\sigma(k)j'_1} + 1$ and $L^{res}_{j'_1} := L^{res}_{j'_1} - l_{\sigma(k)}$. Set $k := k + 1$ and return Step 3.

Now algorithm ILS-LP is formally described as follows. Here, *trial* denotes the current number of iterations of LS-LP from the last improvement, and MAXTRIALS (an input parameter given by users) specifies the upper bound of *trial*. (Π^*, X^*) denotes the best

solution obtained by then. In generating an initial solution (Π^{init}, X^{init}) for LS-LP in each iteration, ILS-LP repeats applying SWAP until a feasible set of cutting patterns is obtained or the terminating condition holds.

Algorithm ILS-LP

Input: Lengths l_i and demands d_i of all products $i \in M$, the number of different cutting patterns n, the length of stock rolls L, and a positive integer MAXTRIALS.

Output: A set of cutting patterns $\Pi^* = \{p_1^*, p_2^*, \ldots, p_n^*\}$ and the numbers of their applications $X^* = \{x_1^*, x_2^*, \ldots, x_n^*\}$

Step 1: Set *trial* := 1. Apply LS-LP to compute (Π, X), and set $(\Pi^*, X^*) := (\Pi, X)$.

Step 2: Choose an initial set of cutting patterns Π^{init} of LS-LP randomly from $N_2^{swap}(\Pi^*)$, and apply SOLVE_IP to compute X^{init}. If the initial solution (Π^{init}, X^{init}) is infeasible, go to Step 4; otherwise go to Step 3.

Step 3: Apply LS-LP to the obtained initial solution (Π^{init}, X^{init}) to compute (Π, X). If $f(\Pi, X) < f(\Pi^*, X^*)$ holds, or $f(\Pi, X) = f(\Pi^*, X^*)$ and $\Delta(\Pi, X) < \Delta(\Pi^*, X^*)$ hold, set $(\Pi^*, X^*) := (\Pi, X)$ and *trial* := 0.

Step 4: If *trial* $\geq$ MAXTRIALS holds, output (Π^*, X^*) and halt; otherwise set *trial* := *trial* + 1 and return to Step 2.

3.7 Computational Experiment

We conducted computational experiment for random instances generated by CUTGEN [32], to compare ILS-LP with other algorithms SHP [46][47], KOMBI [29] and LS-APG (see Chapter 3). We generated 18 classes of random instances by CUTGEN and tested the above algorithms for each class, where the details are the same as in Section 2.6. We coded SHP, LS-APG and ILS-LP in C language and executed on an IBM-compatible personal computer (PentiumIII 1GHz, 1GB memory). The results of KOMBI were taken from [29], as we could not get the source code of KOMBI. KOMBI was run on an IBM-compatible 486/66 personal computer using MODULA-2 as the programming language under MS-DOS 6.0.

We first compare trade-off curves of UNIFORM_INIT and MFF_INIT for an instance as a preliminary experiment, where we take a random instance of class 12 generated by CUTGEN. We tested the following algorithms:

(i) using UNIFORM_INIT and MFF_INIT only,

(ii) starting LS-LP from the initial solutions of UNIFORM_INIT and MFF_INIT,

(iii) starting ILS-LP from the initial solutions generated by UNIFORM_INIT and MFF_INIT.

We apply these algorithms for all n between n_{LB} and m, where

$$n_{LB} = \left\lceil \frac{\sum_{i \in M} l_i}{L} \right\rceil \tag{3.23}$$

is a trivial lower bound of different cutting patterns and m is the number of products. Figure 3.3, 3.4 and 3.5 represent the number of stock rolls f with respect to the number of different cutting patterns n for these algorithms. From Figure 3.3, we observe that the initial solutions of UNIFORM_INIT are better than those of MFF_INIT, but, from Figure 3.4 the solutions of LS-LP using UNIFORM_INIT are much worse than those of LS-LP using MFF_INIT. It shows that the initial solutions generated by UNIFORM_INIT are relatively hard to improve by LS-LP. From Figure 3.5, we observe that the solution of ILS-LP do not much depend on its first initial solution.

We now compare the two local search algorithms LS-LP using UNIFORM_INIT and MFF_INIT to construct an initial feasible solution, where we tested them for 10 instances of 18 classes of random instances. To capture the general behavior of trade-off curves of the above algorithms, we show several points of trade-off curves that attain the minimum numbers of different cutting patterns n satisfying f_{UB} stock rolls or less. Here, the upper bounds of stock rolls f_{UB} are set to as follows:

$$f_{UB} = f^* + \beta f_{LB}, \tag{3.24}$$

where f^* are the numbers of required stock rolls obtained by these algorithms for $n = m$, and f_{LB} is the trivial lower bound of required stock rolls:

$$f_{LB} = \left\lceil \frac{\sum_{i \in M} d_i l_i}{L} \right\rceil. \tag{3.25}$$

We tested these algorithms for $f_{UB} = f^* + \beta f_{LB}$ with $\beta = 0.05, 0.03, 0.01$. Tables 3.2 and 3.3 show the computational results for 18 classes of random instances, where $\bar{n}$ denotes the

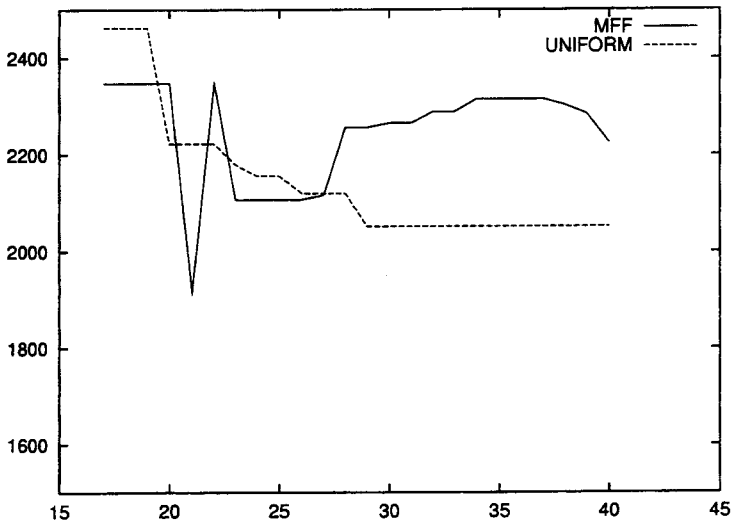

Figure 3.3: (i) Comparison the trade-off curves of UNIFORM_INIT and MFF_INIT

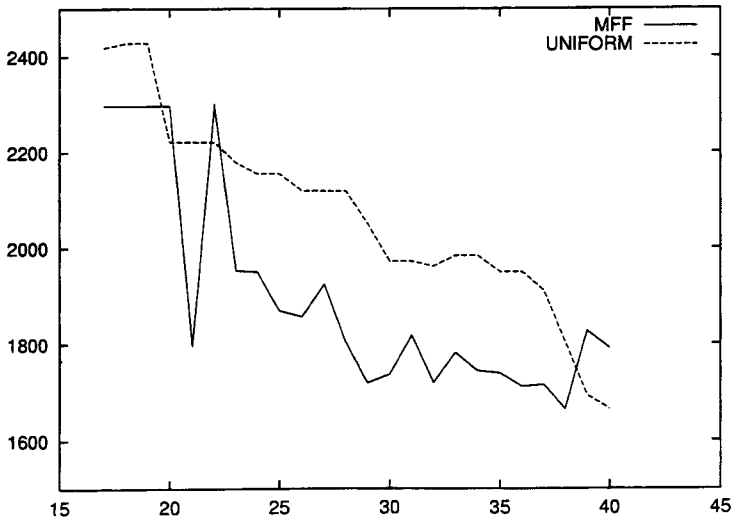

Figure 3.4: (ii) Comparison the trade-off curves of LS-LP using UNIFORM_INIT and MFF_INIT

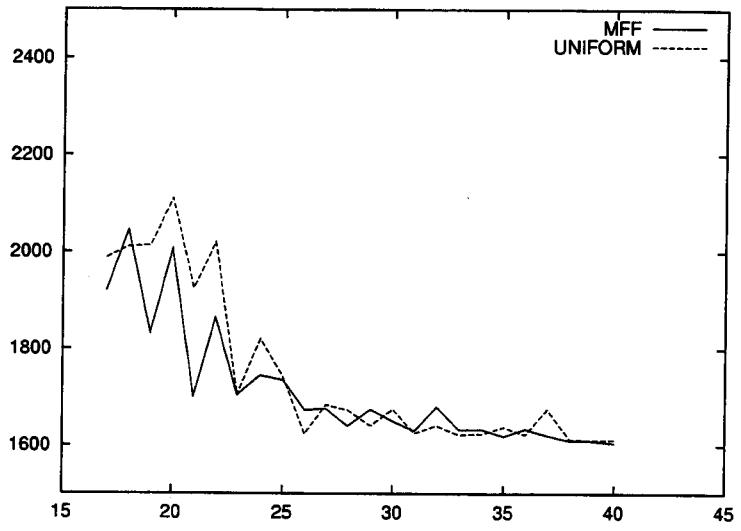

Figure 3.5: (iii) Comparison the trade-off curves of ILS-LP using UNIFORM_INIT and MFF_INIT

average of different cutting patterns, and $\bar{f}$ denotes the average of required stock rolls for each class. From Tables 3.2 and 3.3, it is observed that the numbers of different cutting patterns obtained by LS-LP based on MFF_INIT is much smaller than those obtained by LS-LP based on UNIFORM_INIT, and we conclude that MFF_INIT is more suitable to LS-LP than UNIFORM_INIT.

We compare trade-off curves of ILS-LP (using MFF_INIT) and LS-APG for an instance as a preliminary experiment, where we take a random instance of class 12 generated by CUTGEN. We apply ILS-LP and LS-APG for all n between n_{LB} and m. Figure 3.6 represents the number of stock rolls f with respect to the number of different cutting patterns n for ILS-LP and LS-APG. For this instance, we observe that ILS-LP attains comparable number of stock rolls to LS-APG if the number of different cutting patterns is small. Figure 3.7 shows their CPU time of ILS-LP and LP-APG for different n. We observed that ILS-LP is comparable to LS-APG in its CPU time. From these results, it appears that ILS-LP is comparable to LS-APG while using smaller neighborhood than that of LS-APG.

Table 3.4, 3.5 and 3.6 show the computational results for 18 classes of random instances, where we tested ILS-LP (using MFF_INIT), SHP and KOMBI for 100 instances of each class. The program parameter MAXTL (controlling the upper bound of trim losses for new cutting

Table 3.2: Computational results of LS-LP using UNIFORM_INIT with different f_{UB} for the random instances generated by CUTGEN

			$f^* + 0.05 f_{LB}$		$f^* + 0.03 f_{LB}$		$f^* + 0.01 f_{LB}$	
class	m	$\bar{d}$	$\bar{n}$	$\bar{f}$	$\bar{n}$	$\bar{f}$	$\bar{n}$	$\bar{f}$
1	10	10	3.4	13.3	3.4	13.3	3.4	13.3
2	10	100	5.1	119.5	6.0	117.2	6.7	116.5
3	20	10	8.2	25.4	8.2	25.4	8.2	25.4
4	20	100	7.4	242.5	8.9	238.5	12.0	235.2
5	40	10	14.9	45.7	17.3	44.9	17.3	44.9
6	40	100	14.5	446.0	17.5	438.5	23.5	43.6
7	10	10	7.1	52.7	7.1	52.7	7.1	52.7
8	10	100	7.4	525.6	8.0	521.9	8.2	521.0
9	20	10	15.4	105.4	15.6	105.2	15.8	104.6
10	20	100	15.7	1055.5	15.8	1050.0	16.4	1042.6
11	40	10	29.5	184.5	30.8	182.1	33.6	179.1
12	40	100	31.3	1805.4	32.6	1785.7	35.1	1765.6
13	10	10	8.1	66.8	8.1	66.8	8.1	66.8
14	10	100	8.2	667.5	8.2	667.5	8.6	666.1
15	20	10	15.5	133.0	16.1	132.1	16.5	131.5
16	20	100	15.4	1332.7	15.4	1332.7	16.1	1321.2
17	40	10	34.4	236.1	35.5	233.4	36.5	231.6
18	40	100	35.8	2336.5	36.3	2305.8	37.4	2282.3

Table 3.3: Computational results of LS-LP using MFF_INIT with different f_{UB} for the random instances generated by CUTGEN

			$f^* + 0.05f_{LB}$		$f^* + 0.03f_{LB}$		$f^* + 0.01f_{LB}$	
class	m	$\bar{d}$	$\bar{n}$	$\bar{f}$	$\bar{n}$	$\bar{f}$	$\bar{n}$	$\bar{f}$
1	10	10	2.5	13.6	2.5	13.6	2.5	13.6
2	10	100	4.5	116.5	4.9	115.5	5.2	114.9
3	20	10	4.9	26.6	4.9	26.6	4.9	26.6
4	20	100	7.3	245.3	8.7	241.4	10.0	239.2
5	40	10	8.8	49.7	9.6	48.8	9.6	48.8
6	40	100	11.3	454.7	13.4	447.5	16.3	440.4
7	10	10	6.3	52.5	6.3	52.5	6.3	52.5
8	10	100	7.1	512.8	7.1	512.8	7.5	511.4
9	20	10	11.8	111.8	12.2	110.2	12.2	110.2
10	20	100	13.9	1041.2	15.0	1035.6	16.1	1024.6
11	40	10	19.9	205.3	21.2	203.1	25.2	200.3
12	40	100	27.0	1858.4	30.0	1822.6	31.9	1804.6
13	10	10	7.1	66.9	7.7	66.3	7.7	66.3
14	10	100	7.7	644.5	7.9	641.0	8.0	640.5
15	20	10	13.5	135.6	13.9	135.5	14.9	135.1
16	20	100	14.0	1321.6	14.7	1307.6	15.7	1300.7
17	40	10	26.0	247.0	26.3	246.5	28.1	244.8
18	40	100	30.4	2376.2	31.6	2355.7	32.8	2337.9

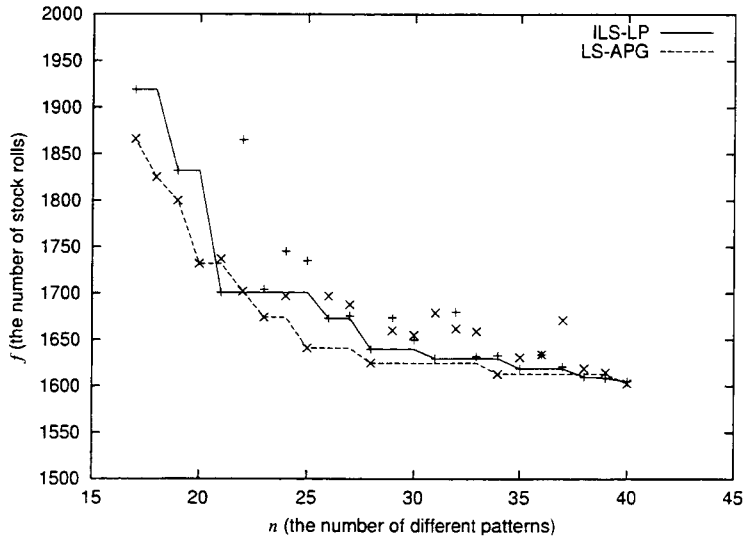

Figure 3.6: The number of stock rolls versus the number of different cutting patterns ($n_{LB} = 17$)

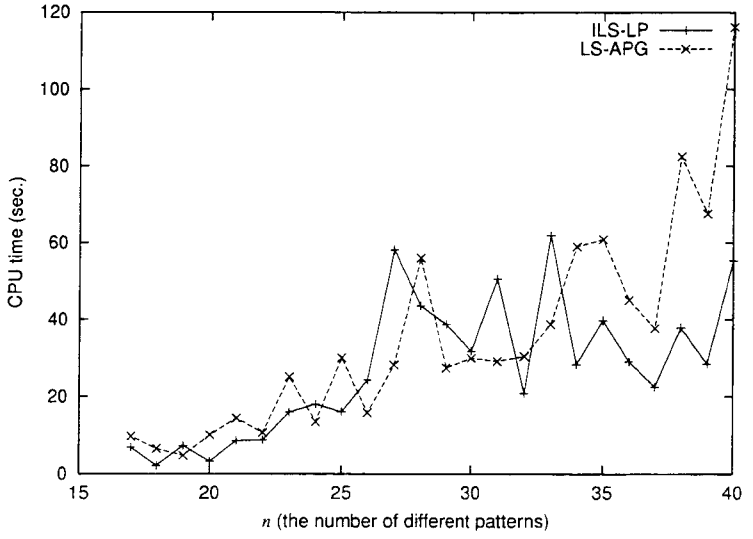

Figure 3.7: The CPU time in seconds versus the number of different cutting patterns (using logarithmic scale for CPU time(sec.))

patterns) of SHP is set to 0.05, 0.03 and 0.01, and MAXTRIALS (the maximum number of iterations of LS-LP) is set to 100. Table 3.4 shows the results of SHP and KOMBI, where SHP was run on three different values of MAXTL= 0.05, 0.03 and 0.01, where Table 3.4 is the same as Table 2.1 in Section 2.6. Tables 3.5 and 3.6 show the results of LS-APG and ILS-LP, respectively, where Table 3.5 is the same as Table 2.2 in Section 2.6. We tested algorithms for $f_{UB} = f^* + \beta f_{LB}$ with $\beta = 0.05, 0.03$ and 0.01.

Table 3.4: Computational results of SHP and KOMBI for the random instances generated by CUTGEN

				SHP						KOMBI	
				MAXTL=0.05		0.03		0.01			
class	m	$\bar{d}$	$\bar{n}_{LB}$	$\bar{n}$	$\bar{f}$	$\bar{n}$	$\bar{f}$	$\bar{n}$	$\bar{f}$	$\bar{n}$	$\bar{f}$
1	10	10	1.67	4.08	11.68	4.25	11.62	4.49	11.57	3.40	11.49
2	10	100	1.67	6.33	112.80	6.33	111.81	6.75	110.85	7.81	110.25
3	20	10	2.56	5.77	22.55	5.89	22.37	5.98	22.17	5.89	22.13
4	20	100	2.56	9.06	220.63	8.98	218.94	9.25	217.00	14.26	215.93
5	40	10	4.26	9.07	43.89	9.03	43.60	9.01	43.17	10.75	42.96
6	40	100	4.26	13.90	434.59	13.45	430.79	13.77	426.81	25.44	424.71
7	10	10	4.62	10.14	52.21	10.33	52.19	10.82	52.77	7.90	50.21
8	10	100	4.62	11.30	519.88	11.46	520.36	11.97	526.81	9.96	499.52
9	20	10	8.65	18.58	97.42	19.22	97.96	19.91	98.82	15.03	93.67
10	20	100	8.65	20.96	970.43	21.74	973.63	21.99	984.32	19.28	932.32
11	40	10	16.27	35.06	186.45	36.29	187.37	37.78	189.91	28.74	176.97
12	40	100	16.27	39.90	1854.79	40.53	1865.41	41.86	1891.90	37.31	1766.20
13	10	10	5.54	10.55	65.46	10.55	65.41	10.67	65.52	8.97	63.27
14	10	100	5.54	10.92	652.95	10.95	654.80	11.12	654.95	10.32	632.12
15	20	10	10.52	20.11	123.36	20.30	124.36	20.86	124.60	16.88	119.93
16	20	100	10.52	21.09	1232.42	21.09	1240.39	21.32	1243.43	19.91	1191.80
17	40	10	19.85	38.02	235.64	38.52	238.06	39.10	239.77	31.46	224.68
18	40	100	19.85	40.16	2351.38	40.38	2366.95	40.67	2389.49	38.28	2242.40

In Tables 3.4, 3.5 and 3.6, we first observe that the numbers of stock rolls attained by KOMBI are smaller than those of other algorithms for all classes. However, other algorithms

Table 3.5: Computational results of LS-APG with different f_{UB} for the random instances generated by CUTGEN

				∞		$f^* + 0.05 f_{LB}$		$f^* + 0.03 f_{LB}$		$f^* + 0.01 f_{LB}$	
class	m	$\bar{d}$	$\bar{n}_{LB}$	$\bar{n}$	$\bar{f}$	$\bar{n}$	$\bar{f}$	$\bar{n}$	$\bar{f}$	$\bar{n}$	$\bar{f}$
1	10	10	1.67	2.00	14.47	2.90	12.54	2.90	12.54	2.90	12.54
2	10	100	1.67	2.00	141.28	4.09	115.32	4.74	113.80	5.73	112.73
3	20	10	2.56	2.57	30.76	4.83	23.66	4.83	23.66	4.83	23.66
4	20	100	2.56	2.57	305.78	6.07	226.04	6.96	223.49	9.28	220.31
5	40	10	4.26	4.28	61.63	8.65	45.72	9.41	45.25	9.41	45.25
6	40	100	4.26	4.28	609.73	9.49	447.94	11.23	441.73	15.23	434.79
7	10	10	4.62	5.01	55.77	6.14	51.45	6.26	51.28	6.26	51.28
8	10	100	4.62	5.01	558.69	6.42	510.60	6.73	507.78	7.38	504.77
9	20	10	8.65	9.27	105.07	10.81	97.39	11.41	96.43	11.81	96.14
10	20	100	8.65	9.27	1053.08	11.44	962.27	12.22	953.92	13.91	944.45
11	40	10	16.27	16.95	201.46	19.60	185.68	20.77	183.79	23.98	181.34
12	40	100	16.27	16.95	2010.26	20.71	1837.90	22.44	1817.07	26.03	1787.39
13	10	10	5.54	6.26	68.73	7.06	64.45	7.30	64.36	7.30	64.36
14	10	100	5.54	6.26	687.14	7.19	644.87	7.45	641.05	7.77	638.53
15	20	10	10.52	11.76	129.10	12.76	123.43	13.33	122.65	14.18	121.81
16	20	100	10.52	11.76	1292.09	13.38	1226.05	14.01	1216.66	15.13	1205.10
17	40	10	19.85	21.50	246.34	22.89	235.85	23.49	233.55	25.58	230.94
18	40	100	19.85	21.50	2471.32	23.99	2334.88	25.35	2305.35	28.35	2274.79

Table 3.6: Computational results of ILS-LP with different f_{UB} for the random instances generated by CUTGEN

				∞		$f^* + 0.05f_{LB}$		$f^* + 0.03f_{LB}$		$f^* + 0.01f_{LB}$	
class	m	$\bar{d}$	$\bar{n}_{LB}$	$\bar{n}$	$\bar{f}$	$\bar{n}$	$\bar{f}$	$\bar{n}$	$\bar{f}$	$\bar{n}$	$\bar{f}$
1	10	10	1.67	1.67	15.99	2.61	12.73	2.61	12.73	2.61	12.73
2	10	100	1.67	1.67	156.73	3.63	114.41	4.06	113.33	5.09	112.14
3	20	10	2.56	2.57	29.49	4.51	24.34	4.51	24.34	4.51	24.34
4	20	100	2.56	2.57	293.16	5.71	225.58	6.50	222.83	8.50	219.97
5	40	10	4.26	4.28	59.26	8.49	47.49	9.22	46.93	9.22	46.93
6	40	100	4.26	4.27	593.21	9.51	446.59	10.80	440.11	14.27	433.97
7	10	10	4.62	5.01	55.70	6.43	51.01	6.69	50.75	6.69	50.75
8	10	100	4.62	5.01	556.84	6.44	507.02	6.74	504.72	7.11	502.96
9	20	10	8.65	9.27	107.24	11.84	96.56	12.50	95.68	13.28	95.13
10	20	100	8.65	9.27	1067.01	11.90	957.01	12.66	947.31	14.23	938.08
11	40	10	16.27	16.95	210.56	23.85	183.74	25.51	181.50	29.68	179.20
12	40	100	16.27	16.95	2123.61	23.19	1823.74	25.14	1798.60	28.11	1777.14
13	10	10	5.54	6.26	68.69	7.48	64.29	7.64	64.13	7.64	64.13
14	10	100	5.54	6.26	685.47	7.33	641.82	7.59	638.97	8.12	635.83
15	20	10	10.52	11.76	130.80	13.75	122.62	14.47	121.71	15.21	121.10
16	20	100	10.52	11.76	1297.19	13.91	1216.75	14.71	1206.47	15.89	1196.96
17	40	10	19.85	21.50	254.11	26.15	234.71	27.58	232.38	29.94	229.67
18	40	100	19.85	21.50	2510.91	26.96	2314.79	28.72	2285.04	31.06	2259.61

can obtain a wide variety of solutions by controlling their input parameters, i.e., they can give trade-off curves between the number of different cutting patterns n and the number of stock rolls f. It is observed that SHP can not reduce the number of different cutting patterns even though it allows cutting patterns having larger trim loss. On the other hand, LS-APG and ILS-LP attain smaller number of different cutting patterns than KOMBI for all classes, without much increasing the number of additional stock rolls. We can also observe that ILS-LP can reduce the number of different cutting patterns effectively as well as LS-APG for all classes. It is necessary to emphasize that ILS-LP introduces basically smaller neighborhood than that LP-APG introduces. As a future topic, we may be able to introduce more powerful neighborhood for ILS-LP. From these observation, we may conclude that ILS-LP can also provide reasonable trade-off curves to some extent, and obtain good solutions comparable to LS-APG while using smaller neighborhood than that of LS-APG.

Table 3.7 shows the average CPU time of SHP (MAXTL=0.03), KOMBI, LS-APG ($f_{UB} = f^* + 0.03f_{LB}$) and ILS-LP ($f_{UB} = f^* + 0.03f_{LB}$), respectively, for all classes. From these instances, SHP is faster than ILS-LP except for class 5 and 6, and KOMBI may be faster than ILS-LP, taking into consideration the power of computers in use. However, the average CPU time of ILS-LP is within 20 seconds except for class 6 and 18, and it may be sufficiently short even if ILS-LP repeatedly applies for all n to obtain a trade-off curve. We also note that ILS-LP can control the trade-off between the quality of solutions and its computational time by the input parameter MAXTRIALS (i.e., the upper bound of iterations of LS-LP).

3.8 Conclusion

We proposed a local search algorithm based on linear programming techniques (LS-LP). It starts from an initial solution obtained by a modified first fit heuristic (MFF) known for the bin packing problem (BPP). Solutions in the neighborhood are restricted to those obtainable by perturbating one cutting pattern in the current set of cutting patterns. In order to find promising directions, we utilize a dual optimal solution of the LP relaxation problem of the auxiliary integer programming problem. Although solutions of LP relaxation is not integer valued, it provides reasonably accurate information as integrality gap is rather small in most instances of 1D-CSP. Since the local search algorithm requires to solve a large number of LP relaxations which are only slightly different each other, we start simplex algorithm from the optimal simplex tableau of the previous solution, instead of starting it from scratch.

Table 3.7: The average CPU time in seconds for the random instances generated by CUTGEN

class	m	$\bar{d}$	SHP	KOMBI	LS-APG	ILS-LP
1	10	10	0.04	0.14	0.01	0.05
2	10	100	0.08	1.14	0.07	0.18
3	20	10	1.56	1.74	0.19	0.66
4	20	100	1.57	16.00	0.76	3.72
5	40	10	631.74	38.03	5.32	17.83
6	40	100	107.11	379.17	11.41	92.40
7	10	10	0.00	0.07	0.03	0.06
8	10	100	0.00	0.20	0.04	0.06
9	20	10	0.01	1.34	0.48	0.80
10	20	100	0.02	3.25	0.71	0.95
11	40	10	0.09	36.27	13.07	18.49
12	40	100	0.14	76.31	19.53	18.80
13	10	10	0.00	0.08	0.03	0.06
14	10	100	0.00	0.13	0.03	0.07
15	20	10	0.01	1.81	0.44	0.87
16	20	100	0.01	2.60	0.60	1.00
17	40	10	0.06	50.93	10.34	14.74
18	40	100	0.10	70.94	14.32	20.55

We modify the simplex algorithm by applying the sensitivity analysis techniques, and apply a variant of the simplex algorithm called the criss-cross algorithm to compute an optimal solution. In order to enhance the local search algorithm, we introduce an iterated local search approach. According to computational experiments, we observed that the iterated local search algorithm based on linear programming techniques (ILS-LP) attains a wide variety of good solutions, and also provides reasonable trade-off curves between the number of different cutting patterns and the number of stock rolls.

Chapter 4

A Variant of 1D-PRP Allowing Underproduction and Overproduction

4.1 Introduction

In this chapter, we consider another formulation of 1D-CSP based on a real application of a chemical fiber industry. As the residual lengths of stock rolls can be easily reused in the chemical fiber industry, minimizing the setup costs for changing cutting patterns is more dominant than minimizing the total trim loss. Furthermore, in some applications, such as the chemical fiber industry, the shortage of demands may be allowed because the additional cost due to the shortage is relatively small. From these observation, we propose a variant of 1D-PRP, called the *quadratic deviation minimization problem* (1D-QDP) which minimizes the amount of quadratic deviation from all demands while using a given number of different cutting patterns.

We propose an iterated local search algorithm based on the *quadratic version of the adaptive pattern generation* (ILS-QAGP), where we have to consider the following three ingredients in the same manner as other proposed algorithms, i.e., (i) how to construct an initial feasible solution, (ii) how to compute auxiliary integer quadratic programming problems (IQP) efficiently, and (iii) how to find promising solutions among all neighbor solutions in the neighborhood. Solutions in the neighborhood are generated by removing one cutting

pattern and adding one new cutting pattern in the current solution. To compute the numbers of applications of the cutting patterns, we propose a heuristic algorithm based on the *nonlinear Gauss-Seidel method* [7]. As it is not realistic to consider all possible feasible cutting patterns, we restrict the candidate cutting patterns to those generated by the quadratic version of adaptive pattern generation (QAPG), where we also indirectly reduce the trim loss by restricting candidate cutting patterns to those having small trim losses. QAPG is also used to generate an initial solution of the local search algorithm.

We conduct computational experiments for random instances generated by CUTGEN [32], and real instances of a chemical fiber industry. ILS-QAPG is compared with other existing heuristic algorithms, e.g., SHP, KOMBI and a heuristic algorithm called the generation and test method (GT) used in the chemical fiber industry. According to the computational results, it is observed that LS-QAPG provides comparable solutions to them.

4.2 Formulation of 1D-QDP

In this section, we define the one dimensional cutting stock problem to minimize the quadratic deviation from demands while using a given number of different cutting patterns n. We are given a sufficient number of stock rolls of length L, and m types of products $M = \{1, 2, \ldots, m\}$ which have given lengths $(l_1, l_2, \ldots, l_m)$ and demands $(d_1, d_2, \ldots, d_m)$. A cutting pattern is described as $p_j = (a_{1j}, a_{2j}, \ldots, a_{mj})$ satisfying

$$\sum_{i \in M} a_{ij} l_i \leq L. \tag{4.1}$$

A solution of 1D-QDP also consists of a set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$, and the numbers of their applications $X = \{x_1, x_2, \ldots, x_n\}$. The 1D-QDP is formulated as follows:

$$\begin{aligned} \text{(1D-QDP)} \quad \text{minimize} \quad & f(\Pi, X) = \sum_{i \in M} \left(\sum_{p_j \in \Pi} a_{ij} x_j - d_i \right)^2 \\ \text{subject to} \quad & \Pi \subseteq S \\ & |\Pi| \leq n \\ & x_j \in \boldsymbol{Z}_+ \ \text{ for all } p_j \in \Pi, \end{aligned} \tag{4.2}$$

where S is the set of all feasible cutting patterns. Notice that minimizing the total trim loss is not equivalent to minimizing the number of stock rolls in 1D-QDP, because shortage of demands is allowed in this problem.

Although the above formulation ignores to minimize the trim loss, if necessary, we can control the quality of trim loss to some extent by applying appropriate constraints on cutting patterns, e.g., restricting S to be the set of complete-cut patterns defined in (1.4). QAPG indirectly reduces the trim loss by restricting the candidate cutting patterns to those having small trim losses.

4.3 Solving Auxiliary Integer Quadratic Programming Problem

For a given set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$, the problem of computing the numbers of their applications $X = \{x_1, x_2, \ldots, x_n\}$ can be described as the following integer quadratic programming problem (IQP):

$$\begin{aligned} (\mathrm{IQP}(\Pi)) \quad & \text{minimize} \quad f(\Pi, X) = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j - d_i \right)^2 \\ & \text{subject to} \quad x_j \in \boldsymbol{Z}_+ \quad \text{for } j = 1, 2, \ldots, n. \end{aligned} \tag{4.3}$$

Since it is hard to solve IQP(Π) exactly, we adopt an approximate solution $\widehat{X} = \{\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n\}$ to IQP(Π) instead of an exact optimal solution. We first solve the quadratic programming (QP) relaxation problem QP(Π) of IQP(Π), in which the integer constraints $x_j \in \boldsymbol{Z}_+$ are replaced with $x_j \geq 0$. After computing an optimal solution $\overline{X} = \{\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\}$ of QP(Π), we round $\overline{x}_j$ to its nearest integer value x_j,

$$x_j := \begin{cases} \lceil \overline{x}_j \rceil & \overline{x}_j - \lfloor \overline{x}_j \rfloor \geq 0.5 \\ \lfloor \overline{x}_j \rfloor & \text{otherwise.} \end{cases} \tag{4.4}$$

There may be several algorithms to solve QP(Π). We use the nonlinear Gauss-Seidel method [7], and abbreviate it NGS, because it is easy to implement and appear to be efficient according to our preliminary computational experiment. In the k-th iteration of NGS, one variable $x_q^{(k)}$ in the current solution $X^{(k)} = \{x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}\}$ is updated as follows. Let $\tilde{x}_q$ be the value of x_q satisfying the equation $\frac{\partial f}{\partial x_q}\Big|_{x_j = x_j^{(k)},\, j \neq q} = 0$. Such $\tilde{x}_q$ can be computed by

$$\tilde{x}_q := x_q^{(k)} - \frac{\frac{\partial f}{\partial x_q}\Big|_{X = X^{(k)}}}{2 \sum_{i \in M} a_{iq}^2}, \tag{4.5}$$

where

$$\left.\frac{\partial f}{\partial x_q}\right|_{X=X^{(k)}} = -2\sum_{i\in M}\lambda_i a_{iq} \tag{4.6}$$

$$\lambda_i = d_i - \sum_{p_j\in\Pi} a_{ij}x_j^{(k)}. \tag{4.7}$$

Then the x_q is updated by $x_q := \max\{0, \tilde{x}_q\}$, and other variables in $X^{(k)}$ are unchanged. Iterations are done so that all variables x_q are scanned in a prespecified order, and all variables x_q are checked in n iterations. NGS is described as follows, where ε is a sufficiently small positive constant.

Algorithm NGS

Input: Demands d_i of all products $i \in M$, and a set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$.

Output: The numbers of applications $X = \{x_1, x_2, \ldots, x_n\}$.

Step 1: Set $\lambda_i := d_i$ for all $i \in M$ and $x_j^{(0)} := 0$, $\left.\frac{\partial f}{\partial x_j}\right|_{X=X^{(0)}} := -2\sum_{i\in M}\lambda_i a_{ij}$ for all $p_j \in \Pi$. Set $k := 0$.

Step 2: If either of the condition (1) $\left|\left.\frac{\partial f}{\partial x_j}\right|_{X=X^{(k)}}\right| < \varepsilon$ holds, or (2) $\left.\frac{\partial f}{\partial x_j}\right|_{X=X^{(k)}} > 0$ and $x_j = 0$ hold, for all $p_j \in \Pi$, then output $X = \{x_1, x_2, \ldots, x_n\}$ and halt.

Step 3: Choose a variable x_q satisfying neither (1) nor (2) in Step 2, and set $\Delta x_q := -\frac{\left.\frac{\partial f}{\partial x_q}\right|_{X=X^{(k)}}}{2\sum_{i\in M} a_{iq}^2}$, and $x_q^{(k+1)} := \max\{0, x_q^{(k)} + \Delta x_q\}$. Set $x_q^{(k+1)} := x_q^{(k)}$ for all $j \neq q$.

Step 4: Update $\lambda_i := \lambda_i - a_{ij}\Delta x_q$ for all $i \in M$, and $\left.\frac{\partial f}{\partial x_j}\right|_{X=X^{(k+1)}} := -2\sum_{i\in M}\lambda_i a_{ij}$ for all $p_j \in \Pi$. Set $k := k + 1$ and return to Step 2.

Finally, we summarize the entire algorithm to compute the numbers of applications $X = \{x_1, x_2, \ldots, x_n\}$ for a given set of cutting pattern $\Pi = \{p_1, p_2, \ldots, p_n\}$.

Algorithm SOLVE_IQP

Input: Demands d_i of all products $i \in M$, and a set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$.

Output: The numbers of applications $X = \{x_1, x_2, \ldots, x_n\}$.

Step 1: Apply NGS to obtain an optimal solution $\overline{X} = \{\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\}$ to QP(Π).

Step 2: For all $p_j \in \Pi$, round $\overline{x}_j$ to the nearest integer:

$$x_j := \begin{cases} \lceil \overline{x}_j \rceil & \overline{x}_j - \lfloor \overline{x}_j \rfloor \geq 0.5 \\ \lfloor \overline{x}_j \rfloor & \text{otherwise.} \end{cases} \tag{4.8}$$

Output $X = \{x_1, x_2, \ldots, x_n\}$ and halt.

4.4 Construction of the Neighborhood

A natural definition of neighborhood $N_1(\Pi)$ is given by replacing one cutting pattern $p_j \in \Pi$ with another new cutting pattern $p'_j \in S \setminus \Pi$:

$$N_1(\Pi) = \{\Pi \cup \{p'_j\} \setminus \{p_j\} \mid p_j \in \Pi, p'_j \in S \setminus \Pi\}, \tag{4.9}$$

where S is the set of all feasible cutting patterns. However, as mentioned in Section 1.5, the number of all feasible cutting patterns $|S|$ is too many to test all of them, and most of these cutting patterns may not lead to improvement. Hence, it is necessary to use much smaller neighborhood which includes only promising solutions. For this purpose, we introduce a small subset $S'(j) \subset S$ for each $p_j \in \Pi$, and define a small neighborhood $N_1^{qapg}(\Pi)$ as follows:

$$N_1^{qapg}(\Pi) = \bigcup_{p_j \in \Pi} N_1^{qapg}(\Pi, j) \tag{4.10}$$

$$N_1^{qapg}(\Pi, j) = \{\Pi \cup \{p'_j\} \setminus \{p_j\} \mid p'_j \in S'(j)\} \text{ for } p_j \in \Pi. \tag{4.11}$$

Now we propose a quadratic version of the adaptive pattern generation algorithm (QAPG), which generates a set of promising cutting patterns $S'(j)$. QAPG is also based the residual demands like other APGs (see Section 2.4), and we utilize the residual demands $r_i(j)$ when a cutting pattern $p_j \in \Pi$ is removed from Π.

$$r_i(j) = \max \left\{ 0, d_i - \sum_{p_q \in \Pi \setminus \{p_j\}} a_{iq} x_q \right\} \text{ for } i \in M. \tag{4.12}$$

We generate a new cutting pattern $p'_j = (a_{ij} \mid i \in M(j))$ by solving the following problem 1D-QDP$_1(j)$:

$$\begin{aligned} (\text{1D-QDP}_1(j)) \quad \text{minimize} \quad & \sum_{i \in M(j)} (a_{ij} x_j - r_i(j))^2 \\ \text{subject to} \quad & \sum_{i \in M(j)} a_{ij} l_i \leq L \\ & a_{ij} \in \boldsymbol{Z}_+ \text{ for all } i \in M(j) \\ & x_j \in \boldsymbol{Z}_+, \end{aligned} \tag{4.13}$$

where $M(j)$ is the subset of M defined as follows:

$$M(j) = \{i \mid r_i(j) > 0\}. \tag{4.14}$$

As 1D-QDP$_1(j)$ contains a kind of the knapsack problem (KP) which is known to be NP-hard [31], we use an approximation algorithm based on the relaxation of 1D-QDP$_1(j)$, i.e., the integer constraints $x_j \in \boldsymbol{Z}_+$ and $a_{ij} \in \boldsymbol{Z}_+$ are replaced with $x_j \geq 0$ and $a_{ij} \geq 0$, respectively. An optimal solution of the relaxation problem is easily found, where it has no trim loss:

$$\begin{aligned} \overline{a}_{ij} &= \left(\frac{L}{\sum_{i \in M(j)} r_i(j) l_i}\right) r_i(j) \quad \text{for all } i \in M(j) \\ \overline{x}_j &= \frac{\sum_{i \in M(j)} r_i(j) l_i}{L}. \end{aligned} \tag{4.15}$$

Hence, we consider to obtain a new cutting pattern by rounding the above solution. The problem of rounding $\overline{a}_{ij}$ to integer value a_{ij} is described as follows:

$$\begin{aligned} (\mathrm{RP}(j)) \quad \text{minimize} \quad & \sum_{i \in M(j)} (a_{ij} - \overline{a}_{ij})^2 \\ \text{subject to} \quad & \sum_{i \in M(j)} a_{ij} l_i \leq L \\ & a_{ij} \in \{\lfloor \overline{a}_{ij} \rfloor, \lceil \overline{a}_{ij} \rceil\} \quad \text{for all } i \in M(j). \end{aligned} \tag{4.16}$$

Since all decision variables a_{ij} can take only two values $\lfloor \overline{a}_{ij} \rfloor$ or $\lceil \overline{a}_{ij} \rceil$, the above problem RP$(j)$ is equivalent to the following 0-1 knapsack problem:

$$\begin{aligned} (\mathrm{RP}'(j)) \quad \text{maximize} \quad & \sum_{i \in M(j)} (1 - 2(\overline{a}_{ij} - \lfloor \overline{a}_{ij} \rfloor)) u_i \\ \text{subject to} \quad & \sum_{i \in M(j)} u_i l_i \leq L - \sum_{i \in M(j)} \lfloor \overline{a}_{ij} \rfloor l_i \\ & u_i \in \{0, 1\} \quad \text{for all } i \in M(j). \end{aligned} \tag{4.17}$$

An optimal solution of RP(j) is obtained from an optimal solution u_i^* of RP$'(j)$ by setting:

$$a_{ij} := \begin{cases} \lfloor \overline{a}_{ij}, \rfloor & u_i^* = 0 \\ \lceil \overline{a}_{ij}, \rceil & u_i^* = 1. \end{cases} \tag{4.18}$$

Taking into account that an optimal solution of RP(j) may not necessarily give a useful cutting pattern, since RP(j) is only an approximation to 1D-QDP$_1(\Pi)$, we try to generate a number of good solutions for RP(j) heuristically. The quadratic version of the adaptive

pattern generation algorithm (QAPG) is based on Sahni's heuristic algorithm [78] for the 0-1 knapsack problem. It first apply a simple greedy algorithm and outputs a solution $(\tilde{u}_i \mid i \in M(j))$. The greedy algorithm starts from $u_i = 0$ for all $i \in M(j)$, and in each step, it chooses the $k \in M(j)$ with the largest $(\overline{a}_{kj} - \lfloor \overline{a}_{kj} \rfloor)/l_k$ among those satisfying:

$$u_k = 0 \text{ and } l_k \leq L - \sum_{i \in M(j)} (\lfloor \overline{a}_{ij} \rfloor + u_i) l_i, \tag{4.19}$$

and then set $u_k := 1$. The algorithm halts when no $k \in M(j)$ satisfies condition (4.19), and outputs the resulting $(u_i \mid i \in M(j))$ as $(\tilde{u}_i \mid i \in M(j))$. After this, the greedy algorithm is repeatedly applied to m problem instances, each of which is obtained by fixing one variable $u_i := 1 - \tilde{u}_i$ for $i \in M(j)$, and consequently $m + 1$ candidate cutting patterns are generated. The greedy algorithm GREEDY and the quadratic version of the adaptive pattern generation algorithm QAPG are described as follows:

Algorithm GREEDY

Input: A set of products $M(j) \subset M$, lengths l_i of all products $i \in M(j)$, an optimal solution $\overline{p}'_j = (\overline{a}_{ij} \mid i \in M(j))$ of the relaxation problem of 1D-QDP$_1(j)$, and the length of stock rolls L.

Output: A set of variables $(u_i \mid i \in M(j))$.

Step 1: Set $u_i := 0$ for all $i \in M(j)$.

Step 2: Sort all products $i \in M(j)$ in the descending order of $(\overline{a}_{ij} - \lfloor \overline{a}_{ij} \rfloor)/l_i$, and let $\sigma(k)$ denote the k-th product in this order.

Step 3: For each $k \in M$, let

$$u_{\sigma(k)} := \begin{cases} 1, & l_{\sigma(k)} \leq L - \sum_{i \in M(j)} (\lfloor \overline{a}_{ij} \rfloor + u_i) l_i \\ 0, & \text{otherwise.} \end{cases} \tag{4.20}$$

Step 4: Output $(u_i \mid i \in M(j))$ and halt.

Algorithm QAPG

Input: A set of products $M(j) \subset M$, lengths l_i of all products $i \in M(j)$, an optimal solution $\overline{p}'_j = (\overline{a}_{ij} \mid i \in M(j))$ of the relaxation problem of 1D-QDP$_1(j)$, and the length of stock rolls L.

Output: $m + 1$ candidate cutting patterns of $p'_j = (a_{ij} \mid i \in M(j))$.

Step 1: Apply GREEDY to obtain a solution $(u_i \mid i \in M(j))$, and set $\tilde{u}_i := u_i$ for all $i \in M(j)$. Set

$$a_{ij} := \begin{cases} \lfloor \overline{a}_{ij}, \rfloor & u_i = 0 \\ \lceil \overline{a}_{ij}, \rceil & u_i = 1, \end{cases} \tag{4.21}$$

and output the cutting pattern $p'_j = (a_{ij} \mid i \in M(j))$. Set $k := 1$.

Step 2: Fix $u_k := \tilde{u}_k$, and apply GREEDY to obtain a solution $(u_i \mid i \in M(j))$. Set a_{ij} for all $i \in M(j)$ according to (4.21), output the cutting pattern $p'_j = (a_{ij} \mid i \in M(j))$. If $k = m$ holds, halt; otherwise $k := k + 1$ and return to Step 2.

To see how many of the cutting patterns generated by QAPG are useful, we conducted computational experiment. We took instances from real applications provided by a chemical fiber industry. The details of these instances are stated in Section 4.6. In this experiment, their sizes are small enough to enumerate all the feasible cutting patterns, and we tested for a solution Π and a cutting pattern $p_j \in \Pi$.

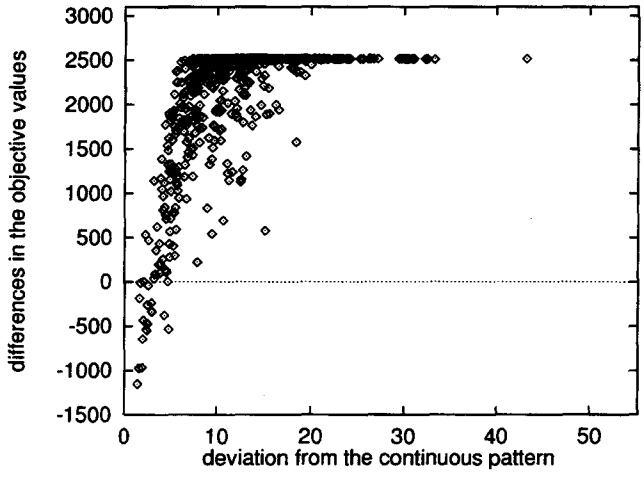

Figure 4.1: Comparison the objective values between Π and $\Pi' \in N_1(\Pi)$

Figure 4.1 represents the difference in the objective values $f(\Pi', X') - f(\Pi, X)$ (vertical axis), where Π' are given by removing the cutting pattern $p_j \in \Pi$ and adding a new cutting pattern $p'_j \in S$. If $f(\Pi', X') - f(\Pi, X) < 0$ holds, the neighbor solution (Π', X') is better than the current solution (Π, X). From Figure 4.1, we observed that the number of neighbor

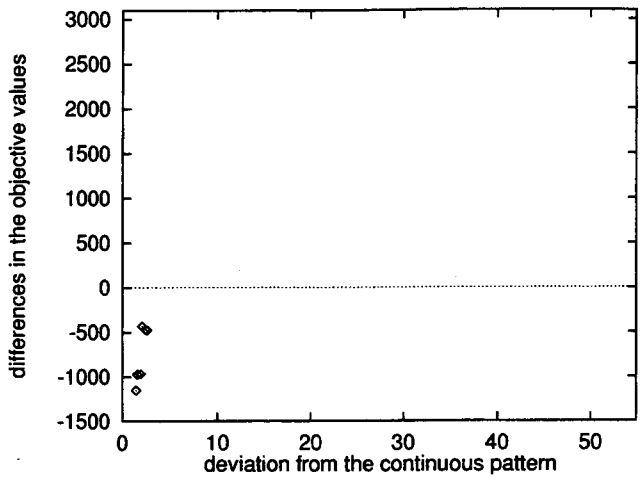

Figure 4.2: Comparison the objective values between Π and $\Pi' \in N_1^{qapg}(\Pi)$

solutions Π' satisfying $f(\Pi', X') - f(\Pi, X) < 0$ is quite small among the neighbor solutions, and that there is a strong correlation between the horizontal and vertical axes in Figure 4.1, i.e., cutting patterns with small deviation from $\overline{a}_{ij}$ are likely to improve the current solution. Next, in Figure 4.2, we show the objective values of solutions $\Pi' \in N_1^{qapg}(\Pi)$ which are generated by removing the cutting pattern $p_j \in \Pi$ and adding a new cutting pattern generated by QAPG. The horizontal and the vertical axes are the same as those in Figure 4.1. Figure 4.2 tells that all cutting patterns p'_j generated by QAPG satisfy $f(\Pi', X') - f(\Pi, X) < 0$. From those results, QAPG appears to be effective to generate good solutions Π' from the current solution Π.

4.5 Entire Algorithm of Local Search

In this section, we explain the framework of the iterated local search algorithm ILS-QAPG. First, we consider a heuristic algorithm to construct an initial solution Π^{init} using QAPG. This algorithm starts from the empty set of cutting patterns $\Pi = \emptyset$, and repeats adding a new cutting pattern p' with the minimum $f(\Pi', X')$ (i.e., $\Pi' = \Pi \cup \{p'\}$) among candidate cutting patterns generated by QAPG. Here, we use the following residual demands r_i instead

of $r_i(j)$ in (4.12):

$$r_i = \max\left\{0, d_i - \sum_{p_j \in \Pi} a_{ij}x_j\right\} \quad \text{for } i \in M. \tag{4.22}$$

Algorithm INIT

Input: Lengths l_i and demands d_i of all products $i \in M$, the number of different cutting patterns n, and the length of stock rolls L.

Output: A set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$, and their numbers of applications $X = \{x_1, x_2, \ldots, x_n\}$.

Step 1: Set $\Pi := \emptyset$ and $k := 0$.

Step 2: Let S' be the set of cutting patterns generated by QAPG with the residual demands r_i for all products $i \in M$. Compute $f(\Pi', X')$ for all $p' \in S'$ by applying SOLVE_IQP, where $\Pi' = \Pi \cup \{p'\}$.

Step 3: Choose the cutting pattern $p' \in S'$ with the minimum $f(\Pi', X')$. Set $\Pi := \Pi \cup \{p'\}$ and $k := k + 1$.

Step 4: If $k = n$ holds, output the solution (Π, X); otherwise return to Step 2.

We now explain the details of the local search algorithm based on the quadratic version of the adaptive pattern generation (LS-QAPG). As noted in Section 4.4, LS-QAPG is based on the small neighborhood $N_1^{qapg}(\Pi)$ constructed by QAPG. Furthermore, to reduce the number of candidate cutting patterns $|S'(j)|$, we check only γ candidate cutting patterns by preferring smaller $\sum_{i \in M(j)}(a_{ij} - \bar{a}_{ij})^2$, where γ is an input parameter. LS-QAPG uses the first admissible move strategy, implemented by a queue that maintains the cutting patterns $p_i \in \Pi$. For the cutting pattern p_j on the top of the queue, we find the cutting pattern p'_j that minimizes $f(\Pi', X')$ among those in $S'(j)$, where $\Pi' = \Pi \cup \{p'_j\} \setminus \{p_j\}$. If $f(\Pi', X') < f(\Pi, X)$ holds, we immediately move to the new solution Π', and put p'_j to the tail of the queue; otherwise we move the cutting pattern p_j from the top to the tail of the queue.

Algorithm LS-QAPG is described as follows. Recall that γ is an input parameter which specifies the number of candidate cutting patterns to be checked. Let Q denote the queue that maintains the cutting patterns $p_j \in \Pi$. The procedure ENQUEUE(Q, p_j) adds the cutting pattern p_j at the tail of Q, TOP(Q) returns the cutting pattern p_j at the top of Q, and DEQUEUE(Q, p_j) removes the cutting pattern p_j from Q.

Algorithm LS-QAPG

Input: Lengths l_i and demands d_i of all products $i \in M$, the number of different cutting patterns n, and the length of stock rolls L. A set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$, and the numbers of their applications $X = \{x_1, x_2, \ldots, x_n\}$.

Output: A set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$, and the numbers of their applications $X = \{x_1, x_2, \ldots, x_n\}$.

Step 1: Set Q be an empty queue, and ENQUEUE(Q, p_j) for all $p_j \in \Pi$ in an arbitrary order. Set $k := 0$.

Step 2: Set $p_j :=$ TOP(Q), and DEQUEUE(Q, p_j). Apply QAPG to generate a set of candidate cutting patterns $S'(j)$. Let $S'(j)$ restrict to the set of cutting patterns which have γ smallest deviations $\sum_{i \in M(j)} (a_{ij} - \overline{a}_{ij})^2$.

Step 3: Compute $f(\Pi', X')$ for all $p'_j \in S'(j)$ by applying SOLVE_IQP, where $\Pi' = \Pi \cup \{p'_j\} \setminus \{p_j\}$. Choose $p'_j \in S'(j)$ with the minimum $f(\Pi', X')$.

Step 4: If $f(\Pi', X') < f(\Pi, X)$ holds, set $(\Pi, X) := (\Pi', X')$, ENQUEUE(Q, p'_j), $k := 0$, and return to Step 2; otherwise ENQUEUE(Q, p_j).

Step 5: Set $k := k+1$. If $k < n$ holds, return to Step 2; otherwise output (Π, X) and halt.

Although we facilitate the neighborhood search by QAPG, the size of neighborhood becomes rather small, and consequently LS-QAPG often converges to a local optimal solution after only a small number of move operations. To overcome such phenomenon, we introduce an extension of local search algorithm called the iterated local search algorithm (ILS). ILS-QAPG starts from LS applied to the initial solution constructed by INIT, and repeats LS from different initial solutions generated by perturbating the best solution obtained by then. The perturbation is done by a random move to a neighbor solution, i.e., it allows a move to a worse solution. Here, let *trial* denote the current number of iterations of LS, and MAXTRIALS (an input parameter given by users) denote the upper bound of *trial*. Let (Π^*, X^*) denote the best solution obtained by then, and (Π, X) denote the current solution.

Algorithm ILS-QAPG

Input: Lengths l_i and demands d_i of all products $i \in M$, the number of different cutting patterns n, and the length of stock rolls L.

Output: A set of cutting patterns $\Pi = \{p_1, p_2, \ldots, p_n\}$, and the numbers of their applications $X = \{x_1, x_2, \ldots, x_n\}$.

Step 1: Set $trial := 1$. Apply INIT to obtain an initial solution (Π, X), and set $(\Pi^*, X^*) := (\Pi, X)$.

Step 2: Apply LS-QAPG to the initial solution to obtain a new local optimal solution (Π, X). If $f(\Pi, X) \leq f(\Pi^*, X^*)$ holds, set $(\Pi^*, X^*) := (\Pi, X)$.

Step 3: If $trial \geq$ MAXTRIALS holds, output (Π^*, X^*) and halt; otherwise randomly choose an initial solution Π of the next LS-QAPG from the neighborhood of the best solution $N_1^{qapg}(\Pi^*)$, set $trial := trial + 1$, and return to Step 2.

4.6 Computational Experiment

We conducted computational experiment for random instances generated by CUTGEN [32], and real instances in a chemical fiber industry. We compared ILS-QAPG with the following three algorithm: SHP [46][47][49], KOMBI [29], and a heuristic algorithm called the generation and test method (GT), which is used in the chemical fiber industry in Japan. GT sequentially adds new cutting patterns step by step, in which, in each step, it generates a set of candidate cutting patterns based on heuristic rules.

We coded ILS-QAPG and SHP in C language and executed on an IBM-compatible personal computer (PentiumII 450MHz, 128MB memory). The results of GT were provided by the chemical fiber industry, where GT was run on an IBM-compatible personal computer (Pentium 133MHz, 32MB memory). The results of KOMBI were taken from [29], as we could not get the source code of KOMBI. KOMBI was run on an IBM-compatible 486/66 personal computer using MODULA-2 as the programming language under MS-DOS 6.0. The program parameter MAXTL of SHP is set to 0.03, and the program parameter γ of ILS-QAPG is set to $\lceil (m+1)/10 \rceil$.

Before presenting computational results, it is necessary to emphasize that the problem solved by SHP and KOMBI is different from that solved by ILS-QAPG and GT, because demand constraints are treated differently. ILS-QAPG and GT allow the shortage and/or overproduction of the products, where ILS-QAPG minimizes their total quadratic deviation while using a given number of different cutting patterns, and GT reduces both their total

deviation and the number of different cutting patterns heuristically. KOMBI do not allow the shortage, and SHP allows neither the shortage nor overproduction. Therefore, precise comparison of these algorithms is not possible. However, we may be able to capture their general tendency from the computational results.

Our primary objective is to attain sufficiently small deviation while using a given number of different cutting patterns. To evaluate the quality, it may be convenient to introduce a simple criterion of goodness. Let us consider $f(\Pi, X)$ be *acceptable* (i.e., sufficiently small) if $f(\Pi, X) \leq b_{acpt}$ holds, where

$$b_{acpt} = \sum_{i \in M} \max\{(0.01 d_i)^2, 1.0\}, \tag{4.23}$$

i.e., 1% of the demand or a single deviation for each product $i \in M$. In the following results, the number of acceptable instances n_{acpt} is always given for ILS-QAPG.

We first conducted computational experiment for the random instances generated by CUTGEN, and compared ILS-QAPG with SHP and KOMBI. We used 18 classes of random instances by CUTGEN like in Sections 2.6 and 3.7, where the details of these instances are described in Section 2.6. For each class, 10 instances were generated and solved by SHP and ILS-QAPG, and 100 instances were solved by KOMBI. Table 4.1 shows the results of SHP, KOMBI and ILS-QAPG, where *tloss* is the ratio (percentage) of the total trim loss to the length of stock rolls:

$$tloss = \frac{100 \sum_{p_j \in \Pi} (L - \sum_{i \in M} a_{ij} l_i)}{L \sum_{p_j \in \Pi} x_j}. \tag{4.24}$$

Note that $|\Pi|$ and *tloss* are averaged over 10 instances for SHP and ILS-QAPG, while they are averaged over 100 instances for KOMBI (the data are taken form [29]). As the primal goal of this experiment was to test the performance of ILS-QAPG with small number of different cutting patterns n, we ran ILS-QAPG for six cases $n = \alpha, \alpha - 1, \alpha - 2$ and $n = \beta, \beta - 1, \beta - 2$. The parameters α and β are defined as follows:

$$\alpha = |\Pi_{\mathrm{SHP}}| \tag{4.25}$$

$$\beta = \left\{ |\Pi_{\mathrm{SHP}}|, \left\lceil \overline{|\Pi_{\mathrm{SHP}}|} - \overline{|\Pi_{\mathrm{KOMBI}}|} \right\rceil \right\}, \tag{4.26}$$

where $|\Pi_{\mathrm{SHP}}|$ denotes the number of different cutting patterns obtained by SHP, $\overline{|\Pi_{\mathrm{SHP}}|}$ denotes the average of $|\Pi_{\mathrm{SHP}}|$ for the corresponding instances, and $\overline{|\Pi_{\mathrm{KOMBI}}|}$ denotes the average of different cutting patterns obtained by KOMBI. That is, β is intended to represent the smaller

of $|\Pi_{\text{SHP}}|$ and $|\Pi_{\text{KOMBI}}|$ approximately (recall that $|\Pi_{\text{KOMBI}}|$ for individual instance is not given in [29]).

Table 4.1: Computational results of SHP, KOMBI and ILS-QAPG for the random instances

							ILS-QAPG					
			SHP		KOMBI		$\|\Pi\| = \beta$		$\|\Pi\| = \beta - 1$		$\|\Pi\| = \beta - 2$	
class	m	$\bar{d}$	$\|\Pi\|$	*tloss*	$\|\Pi\|$	*tloss*	n_{acpt}	*tloss*	n_{acpt}	*tloss*	n_{acpt}	*tloss*
1	10	10	3.9	5.09	3.40	4.47	4	2.74	9	2.81	2/6*	2.51
2	10	100	5.5	1.83	7.81	0.47	9	2.67	7	2.90	9	2.73
3	20	10	6.1	3.42	5.89	2.52	9	0.96	9	0.96	5	1.83
4	20	100	8.4	1.20	14.26	0.25	7	1.37	5	1.17	7	1.14
5	40	10	9.3	3.04	10.75	1.10	10	0.71	10	0.61	10	0.67
6	40	100	13.1	1.57	25.44	0.12	6	0.72	6	0.70	4	0.69
7	10	10	10.3	16.78	7.90	15.41	9	15.01	7	14.06	5	13.87
8	10	100	11.9	16.58	11.9	9.96	9	16.09	8	17.11	6	15.54
9	20	10	18.9	15.12	18.9	15.03	9	11.45	9	12.06	9	13.19
10	20	100	21.7	15.61	19.28	10.72	10	11.86	9	11.72	10	12.08
11	40	10	37.6	11.93	28.74	7.33	8	4.97	8	4.88	10	7.52
12	40	100	41.2	11.15	37.31	7.29	7	6.37	7	6.00	9	6.77
13	10	10	10.8	18.66	8.97	19.17	9	18.23	8	17.08	5	17.74
14	10	100	11.2	18.48	10.32	18.55	9	18.16	6	18.05	4	17.66
15	20	10	19.5	17.48	16.88	14.76	10	16.69	10	16.74	9	16.45
16	20	100	21.3	18.00	19.91	14.67	9	17.88	9	16.93	5	17.22
17	40	10	37.7	14.33	31.46	10.30	10	10.10	10	10.00	10	9.69
18	40	100	40.7	14.45	38.28	10.22	10	10.32	10	10.76	10	10.36

* only 6 instances are executable.

From Table 4.1, we observe that ILS-QAPG obtains acceptable solutions in many cases while using smaller number of different cutting patterns than SHP and KOMBI. This may indicate that the primal goal of ILS-QAPG is achieved. Table 4.1 also shows that *tloss* of ILS-QAPG is smaller than that of SHP in 17 classes, and is smaller than that of KOMBI in 9 classes. Although ILS-QAPG has weaker constraint than SHP and KOMBI, these results indicate that the trim loss of ILS-QAPG is sufficiently small for practical purposes.

Table 4.2 shows the CPU time of SHP, KOMBI and ILS-QAPG (with $|\Pi| = \beta$) for random

Table 4.2: CPU time in seconds of SHP, KOMBI and ILS-QAPG for random instances

class	m	$\bar{d}$	SHP	KOMBI	ILS-QAPG
1	10	10	0.09	0.14	0.07
2	10	100	0.11	1.14	0.57
3	20	10	2.28	1.74	0.38
4	20	100	2.71	16.00	2.89
5	40	10	180.10	38.03	3.25
6	40	100	256.58	379.17	20.85
7	10	10	0.01	0.07	0.20
8	10	100	0.02	0.20	0.86
9	20	10	0.04	1.34	1.54
10	20	100	0.06	3.25	9.46
11	40	10	0.22	36.27	25.14
12	40	100	0.32	76.31	318.45
13	10	10	0.01	0.08	0.15
14	10	100	0.02	0.13	0.33
15	20	10	0.03	1.81	0.85
16	20	100	0.04	2.60	2.52
17	40	10	0.16	50.93	4.87
18	40	100	0.24	70.94	40.75

instances, respectively. For these classes of instances, SHP and KOMBI are faster than ILS-QAPG except for 2 classes (recall that KOMBI was run on a slower personal computer). In summary, for random instances, we may conclude that, ILS-QAPG tends to produce solutions of better quality in the sense of smaller numbers of different cutting patterns and small trim loss, at the cost of consuming more computational time.

We next conducted computational experiments for real instances provided by a chemical fiber industry. The data of these are available at our world wide web site[1]. There are 40 instances with m ranging from 6 to 29, $L = 9080, 5180$, d_i ranging from 2 to 264, and l_i ranging from 500 to 2000. Table 4.3 (resp., Table 4.4) shows the results of SHP, GT and ILS-QAPG for the instances with $L = 9080$ (resp., $L = 5180$).

Table 4.3 (resp., Table 4.4) tells that 20/20 (resp., 19/20) are acceptable for $|\Pi| = \alpha$, 16/20 (resp., 17/20) are acceptable for $|\Pi| = \alpha - 1$, and 10/20 (resp., 15/20) are acceptable for $|\Pi| = \alpha - 2$. GT also gives acceptable solutions in 19/20 instances (it fails to obtain a feasible solution for one instance). SHP is designed so that a solution with $f = 0$ is output (i.e., always acceptable). However, SHP and GT achieve this performance at the cost of using larger number of different cutting patterns in most cases, as observed in Tables 4.3 and 4.4. If we compare ILS-QAPG with SHP and GT from the view point of the obtained number of different cutting patterns, SHP outperforms GT in Table 4.3, but the relation is reversed in Table 4.4. This suggests that SHP performs well for instances in which the ratio of product lengths l_i to the length of stock rolls L is relatively small, but not so if the ratio is relatively large. If we evaluate the quality of solutions from *tloss* (trim loss), SHP has smaller *tloss* than those of ILS-QAPG and GT. The performance of SHP is remarkable in this respect, but ILS-QAPG also performs reasonably well (considering that the minimization of the trim loss is not a primal target of ILS-QAPG). It is worth mentioning that ILS-QAPG achieve almost the same *tloss* even if smaller number of different cutting patterns are used; its performance is robust is the sense of *tloss*.

Table 4.5 shows the CPU time of SHP, GT and ILS-QAPG with $|\Pi| = \alpha$ for real instances, respectively. The CPU time of SHP becomes extremely large for some instances, because SHP generates a large number of candidate cutting patterns for such instances. Similar tendency is observed for GT. The CPU time of ILS-QAPG is comparable to other two algorithms, and appears to be more stable.

[1] `http://www-or.amp.i.kyoto-u.ac.jp/members/umetani/data.html`

Table 4.3: Computational results of SHP, GT and ILS-QAPG for real instances ($L = 9080$)

		SHP			GT			ILS-QAPG					
								$\|\Pi\| = \alpha$		$\|\Pi\| = \alpha - 1$		$\|\Pi\| = \alpha - 2$	
m	b_{acpt}	$\|\Pi\|$	f	*tloss*	$\|\Pi\|$	f	*tloss*	f	*tloss*	f	*tloss*	f	*tloss*
6	6.00	5	0	2.95	3	1	3.61	1	3.67	13	3.27	236	10.62
7	7.00	4	0	5.62	3	9	1.58	1	0.99	1	0.99	15	1.00
8	13.97	4	0	1.02	6	1	0.43	4	4.71	7	4.73	34	4.73
9	9.96	5	0	2.78	6	10	0.76	2	6.75	2	6.86	21	6.39
10	10.82	5	0	1.73	6	4	1.17	4	2.18	23	2.15	97	2.19
11	11.69	5	0	2.33	7	4	1.75	8	2.24	23	3.00	10	6.31
13	13.00	6	0	2.52	7	2	1.87	2	7.52	4	6.97	4	6.17
13	13.00	4	0	2.99	6	0	2.99	2	4.14	3	3.95	30	4.14
14	14.00	5	0	5.33	5	13	3.01	5	0.63	17	1.03	105	0.83
15	16.07	5	0	1.29	7	19	2.12	7	1.81	11	0.95	40	1.78
16	16.84	6	0	2.87	8	3	7.02	2	2.88	7	2.69	19	2.44
17	17.12	12	0	1.02	8	5	6.98	0	3.08	6	3.31	3	2.94
18	18.00	6	0	2.29	10	5	10.77	8	2.73	12	2.65	43	2.14
19	21.59	8	0	2.33	10	9	4.04	3	2.52	3	2.76	17	2.69
20	20.00	9	0	4.69	–	–	–	2	3.35	3	3.44	4	1.72
23	25.92	8	0	2.58	11	0	6.02	18	3.14	17	3.91	15	3.64
26	38.75	9	0	1.91	13	0	14.39	23	2.78	15	2.82	16	1.73
28	28.00	8	0	3.42	14	18	6.31	8	3.52	7	3.78	25	2.77
28	29.46	12	0	2.36	14	5	7.79	1	2.15	2	2.04	1	2.15
29	29.00	13	0	2.95	10	9	11.39	2	2.38	3	2.02	4	1.75

Table 4.4: Computational results of SHP, GT and ILS-QAPG for real instances ($L = 5180$)

		SHP			GT			ILS-QAPG					
								$\|\Pi\| = \alpha$		$\|\Pi\| = \alpha - 1$		$\|\Pi\| = \alpha - 2$	
m	b_{acpt}	$\|\Pi\|$	f	$tloss$	$\|\Pi\|$	f	$tloss$	f	$tloss$	f	$tloss$	f	$tloss$
6	6.00	7	0	4.93	4	2	5.01	1	5.22	3	4.65	103	4.35
7	7.00	10	0	7.55	6	4	3.69	0	7.55	3	7.16	3	7.17
8	13.97	10	0	3.16	4	1	3.43	16	7.72	138	3.57	179	3.82
9	9.96	9	0	6.75	6	3	8.37	1	8.78	3	8.66	6	8.53
10	10.82	14	0	4.03	8	1	3.71	3	3.91	1	7.71	3	7.79
11	11.69	12	0	2.89	8	1	4.10	1	4.61	2	5.57	4	5.12
13	13.00	14	0	2.36	6	10	2.07	9	2.05	25	3.83	106	2.36
13	13.00	9	0	2.83	7	1	3.29	1	3.20	1	3.20	1	3.20
14	14.00	11	0	3.20	6	1	3.60	1	5.56	6	3.72	37	5.54
15	16.07	16	0	2.87	7	2	4.37	3	4.83	2	5.04	2	4.37
16	16.84	10	0	2.71	8	1	2.42	3	1.92	19	4.15	38	3.05
17	17.12	9	0	2.92	9	9	3.34	2	3.27	2	3.36	2	3.27
18	18.00	11	0	1.88	11	14	2.51	5	4.07	1	5.61	17	4.06
19	21.59	25	0	4.75	12	5	5.05	3	5.34	1	5.59	2	5.02
20	20.00	8	0	3.81	8	2	2.63	1	3.52	4	2.33	9	5.74
23	25.92	15	0	1.39	11	9	4.72	2	5.09	5	5.15	14	4.86
26	38.75	29	0	2.26	16	1	3.23	0	4.77	4	4.62	1	4.40
28	28.00	11	0	1.25	13	10	1.73	5	3.81	12	3.89	14	3.36
28	29.46	15	0	1.37	12	5	2.58	7	4.51	5	4.74	16	4.72
29	29.00	13	0	1.22	13	4	8.20	2	5.33	2	5.48	2	5.23

Table 4.5: The CPU time in seconds for real instances ($L = 9080, 5180$)

	$L = 9080$			$L = 5180$		
m	SHP	GT	ILS-QAPG	SHP	GT	ILS-QAPG
6	0.02	0.22	0.03	0.01	0.11	0.09
7	0.01	0.82	0.28	0.09	0.27	0.23
8	0.02	1.32	0.10	0.07	0.22	0.20
9	0.02	0.93	0.41	0.10	1.65	0.43
10	0.08	1.54	0.35	0.20	0.44	0.59
11	0.05	2.42	0.41	0.30	0.99	1.00
13	0.17	3.62	0.68	0.20	2.41	0.45
13	0.30	2.85	0.23	0.06	0.88	0.32
14	0.16	1.54	0.32	0.07	1.92	0.31
15	0.13	2.47	0.73	1.54	1.04	0.46
16	0.17	9.11	0.56	0.04	6.86	0.98
17	46.62	3.57	1.52	0.06	3.85	1.46
18	0.22	26.97	0.48	0.05	2.85	1.93
19	2.33	3.24	2.06	2.37	5.55	3.27
20	4.09	-	0.68	0.03	10.87	0.46
23	0.40	36.03	2.64	0.17	12.80	2.70
26	1.27	7.80	5.06	2.26	15.92	14.52
28	4.80	596.99	2.62	0.27	20.87	3.88
28	197.74	25.70	8.09	0.49	9.17	5.08
29	597.02	78.27	2.37	0.23	6.15	4.42

4.7 Reduction of Computational Time

To understand the rapid growth of CPU time of ILS-QAPG with m in Table 4.2, we conducted additional computational experiment. That is, for the instances generated by CUTGEN with $\bar{d} = 100$, $(\nu_1, \nu_2) = (0.01, 0.8)$ and $m = 10, 15, 20, 30, 40$, we applied the simple local search algorithm LS-QAPG from initial solutions generated by INIT. LS-QAPG was applied to 10 instances for each m, in which two cases of $\gamma = m + 1$ and $\gamma = 5$ were tested (recall that γ is a program parameter which restricts the number of candidate cutting patterns scanned). Here, the number of different cutting patterns $|\Pi|$ is set to $|\Pi_{\mathrm{SHP}}| - 2$. Table 4.6 gives the results of this experiment, where

$\bar{n}$: the average of different cutting patterns $|\Pi|$ ($|\Pi| = |\Pi_{\mathrm{SHP}}| - 2$).

$\#f$: the average of evaluations of $f(\Pi, X)$ in one execution of LS-QAPG (i.e., the number of calls to NGS).

#loops : the average of iterations of the loop (Step 2–4) in one execution of NGS.

#moves : the average of moves in one execution of LS-QAPG.

CPU time : the average of CPU time in one execution of LS-QAPG in seconds.

Table 4.6: Performance of LS-QAPG for random instances

γ	m	$\bar{n}$	$\#f$	#loops	#moves	CPU time	n_{acpt}	*tloss*
	10	9.9	179.5	32.57	2.0	0.0953	8	16.13
	15	14.7	352.3	38.96	2.4	0.468	7	11.24
$m+1$	20	19.7	974.8	34.65	7.7	1.91	6	11.17
	30	28.7	2098.0	49.33	10.4	11.4	6	5.54
	40	39.2	4014.9	65.54	17.4	51.4	7	5.47
	10	9.9	92.6	29.27	2.3	0.0455	8	15.82
	15	14.7	175.6	23.95	4.8	0.145	9	11.72
5	20	19.7	265.8	31.12	7.3	0.459	6	11.12
	30	28.7	397.3	35.04	5.7	1.50	6	5.25
	40	39.2	553.8	61.41	11.4	6.65	6	5.19

From Table 4.6, we see that $\bar{n}$, $\#f$, #loops and #moves are approximately proportional to $m^{0.99}$, $m^{2.24}$, $m^{0.50}$ and $m^{1.56}$, respectively. As the time to execute one loop of NGS can be estimated as $O(mn)$, this tells that the average CPU time of one execution of LS-QAPG is roughly given by

$$\#f \cdot \#loops \cdot O(m\bar{n}) = O(m^{4.74}), \tag{4.27}$$

which may be justified by the column of CPU time (i.e., proportional to $m^{4.54}$).

These observations suggest that, in order to prevent the rapid growth of the CPU time with m, it is important

(i) to reduce the size of neighborhood without sacrificing the power of the local search algorithm,

(ii) to improve the Gauss-Seidel method (or to use other methods) so that #loops and the time for one loop can be reduced.

Although the point (ii) still remains to be a topic of future research, we tried point (i) by controlling the parameter γ in Step 2 of LS-QAPG (recall that γ restricts the number of cutting patterns scanned). For example, in all computational experiments of Section 4.7, we used $\gamma = \lceil (m+1)/10 \rceil$ instead of $\gamma = m + 1$. This modification reduced the CPU time to about 1/10 of that of Table 4.6, almost without sacrificing the power of local search algorithm. Table 4.6 also contains the results with $\gamma = 5$ (i.e., constant). In this case, $\#f$ decreases to $m^{1.29}$ from $m^{2.24}$, the average CPU time of one execution of LS-QAPG decreases to $m^{3.60}$ from $m^{4.54}$. This modification still does not appear to sacrifice the power of local search algorithm much.

4.8 Conclusion

We considered in this chapter another formulation of 1D-CSP based on a real application of a chemical fiber industry, called the quadratic deviation minimization problem (1D-QDP), in which both underproduction and overproduction of products are allowed. For this problem, we proposed an iterated local search algorithm based on the quadratic version of the adaptive pattern generation (ILS-QAPG). ILS-QAPG tries to find a set of n cutting patterns yielding sufficiently small deviation from demands. Solutions in the neighborhood is obtained by removing one cutting pattern in the current set and adding one new cutting pattern in the

candidate list. To generate only promising solutions in the neighborhood search, we introduce the quadratic version of the adaptive pattern generation (QAPG) which generates new cutting patterns based on the residual demands when one cutting pattern is removed. We conducted computational experiments for random instances generated by CUTGEN and real instances in a chemical fiber industry, and observed that the performance of ILS-QAPG is comparable to other existing heuristic algorithm SHP, KOMBI and GT (which solve a slightly different formulation of 1D-CSP).

Chapter 5

Conclusion

Throughout this thesis, we considered several mathematical models and metaheuristic algorithms for one dimensional cutting stock problems (1D-CSP). The contribution of this thesis is summarized as follows.

First, we proposed a new formulation by considering the number of different cutting patterns n as an input parameter given by users. We call this variant of 1D-CSP as the pattern restricted version of 1D-CSP (1D-PRP), which minimizes the number of stock rolls. By solving 1D-PRP for different parameter values n, we can obtain trade-off curves between the different number of cutting patterns and the number of stock rolls. Using this, we can make more careful analyses of these objective functions, and give more desirable solutions according to requirements of users.

Second, for this problem 1D-PRP, we proposed a local search algorithm based on an adaptive pattern generation (LS-APG). It starts from an initial solution obtained by a modified first fit heuristic (MFF) known for the bin packing problem (BPP). Solutions in the neighborhood are defined by removing two cutting patterns from the current solution and adding two new cutting patterns from the set of candidate cutting patterns. However, the number of all feasible cutting patterns is too large to evaluate all of them, since it grows exponentially in the number of products. To facilitate the search in the neighborhood, we introduced the adaptive pattern generation (APG) to construct a small subset of the neighborhood containing good solutions. The adaptive pattern generation is based on the residual demands when two cutting patterns are removed from the current solution. From computational experiments for random instances, we observed that LS-APG attains a wide variety of

good solutions comparable to SHP and KOMBI, and LS-APG provides reasonable trade-off curves between the number of different cutting patterns and the number of stock rolls over a very wide range.

Next, we proposed another local search algorithm based on linear programming techniques (LS-LP). It starts from an initial solution obtained by a modified first fit heuristic (MFF) other than that of LS-APG. Solutions in the neighborhood are restricted to those obtainable by perturbating one cutting pattern in the current set of cutting patterns. In order to find promising directions, we utilize a dual optimal solution of LP relaxation problem of the auxiliary integer programming problem. Although solutions of LP relaxation is not integer valued, they provide reasonably accurate information as these integrality gaps are rather small in most instances of 1D-CSP. Since the local search algorithm requires to solve a large number of LP relaxations which are only slightly different each other, we start the simplex algorithm from the optimal simplex tableau of the previous solution, instead of starting it from scratch. We modify the simplex algorithm by applying sensitivity analysis techniques, and apply a variant of the simplex algorithm called the criss-cross algorithm to compute an optimal solution. In order to enhance the local search algorithm, we introduce an iterated local search approach. According to computational experiments for random instances, we observed that the iterated local search algorithm based on linear programming techniques (ILS-LP) obtains a wide variety of good solutions, and provides reasonable trade-off curves between the number of different cutting patterns and the number of stock rolls, similarly to LS-APG.

Finally, we considered another formulation of 1D-CSP on a real application of a chemical fiber industry, called the quadratic deviation minimization problem (1D-QDP), in which both underproduction and overproduction of products are allowed. For this problem, we proposes an iterated local search algorithm based on the quadratic version of the adaptive pattern generation (ILS-QAPG). It tries to find a set of n cutting patterns yielding sufficiently small deviation from demands. Solutions in the neighborhood is obtained by removing one cutting pattern in the current set and adding one new cutting pattern in the candidate list. To generate only promising solutions in the neighborhood search, we introduce the quadratic version of the adaptive pattern generation (QAPG) which generates new cutting patterns based on the residual demands when one cutting pattern is removed. We conducted computational experiments for random instances and real instances in a chemical fiber industry,

and observed that the performance of ILS-QAPG is comparable to other existing heuristic algorithm SHP, KOMBI and GT (which solve a slightly different formulation of 1D-CSP).

In recent years, as most of systems in real applications have become more sophisticated, problems have become more complicated than those simple local search and metaheuristic algorithms can handle. In order to cope with these phenomena, many hybrid algorithms have been studied, i.e., various heuristic algorithms and exact algorithms are introduced to local search algorithms and metaheuristic algorithms, or basic principle of metaheuristic algorithms are combined together to make more powerful tools solving these intractable problems. However, we must take into account the fact that these hybridization of algorithms may often spoil the flexibility and simplicity of local search and metaheuristic algorithms. The author hopes that this thesis will provide some assistance to the community of metaheuristic algorithms.

Bibliography

[1] E.H.L. Aarts and J.K. Lenstra (eds.), *Local Search in Combinatorial Optimization*, John Wiley and Sons (1997).

[2] E.H.L. Aarts, P.J.M. Laarhoven, C.L. Liu and P. Pan, "VLSI layout synthesis," in: *Local Search in Combinatorial Optimization*, E.H.L. Aarts and J.K. Lenstra (eds.), John Wiley and Sons (1997) 415–440.

[3] R.K. Ahuja, J.B. Orlin and D. Sharma, "Very large-scale neighborhood search," *International Transactions in Operational Research* 7 (2000) 301–317.

[4] E.J. Anderson, C.A. Glass and C.N. Potts, "Machine scheduling," in: *Local Search in Combinatorial Optimization*, E.H.L. Aarts and J.K. Lenstra (eds.), John Wiley and Sons (1997) 361–397.

[5] J.E. Beasley, "An algorithm for solving large capacitated warehouse location problems," *European Journal of Operational Research* 33 (1988) 314–325.

[6] J.E. Beasley, "Lagrangian heuristics for location problems," *European Journal of Operational Research* 65 (1993) 383–399.

[7] D.P. Bertsekas, *Nonlinear Programming*, Athena Scientific (1995).

[8] P. Brucker, *Scheduling Algorithms*, Springer-Verlag (1995).

[9] B. Cao and F. Glover, "Tabu search and ejection chains — application to a node weighted version of the cardinality-constrained TSP," *Management Science* 43 (1997) 908–921.

[10] A. Caprara, M. Fischetti and P. Toth, "A heuristic method for the set covering problem," *Operations Research* 47 (1999) 730–743.

[11] S. Ceria, P. Nobili and A. Sassano, "A Lagrangian-based heuristic for large-scale set covering problems," *Mathematical Programming* 81 (1998) 215–228.

[12] M. Chams, A. Hertz and D. de Werra, "Some experiments with simulated annealing for coloring graphs," *European Journal of Operational Research* 32 (1987) 260–266.

[13] I. Charon and O. Hudry, "The noising method: a new method for combinatorial optimization," *Operations Research Letters* 14 (1993) 133–137.

[14] C.H. Cheng, B.R. Feiring and T.C.E. Cheng, "The cutting stock problem — a survey," *International Journal of Production Economics* 36 (1994) 291–305.

[15] K.L. Clarkson, "A Las Vegas algorithm for linear and integer programming when the dimension is small," *Journal of the ACM* 42(2) 488–499.

[16] B. Codenotti, G. Manzini, L. Margara and G. Resta, "Perturbation: an efficient technique for the solution of very large instances of the Euclidean TSP," *INFORMS Journal of Computing* 8 (1996) 125–133.

[17] G.B. Dantzig, *Linear Programming and Extensions*, Princeton University Press (1963).

[18] Z. Degraeve and L. Schrage, "Optimal integer solutions to industrial cutting stock problems," *INFORMS Journal of Computing* 11(4) (1999) 406–419.

[19] M. Dorigo and L.M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation* 1 (1997) 53–66.

[20] A. Drexl and A. Kimms, "Lot sizing and scheduling — survey and extensions," *European Journal of Operational Research* 99 (1997) 221–235.

[21] G. Dueck and T.Scheuer, "Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing," *Journal of Computational Physics* 90 (1990) 161–175.

[22] G. Dueck, "New optimization heuristics: the great deluge algorithm and record-to-record travel," *Journal of Computational Physics* 104 (1993) 86–92.

[23] H. Dychoff, "A typology of cutting and packing problems," *European Journal of Operational Research* 44 (1990) 145–159.

[24] H. Dyckhoff, G. Scheithauer and J. Terno, "Cutting and packing," in: M. Dell'Amico, F. Maffioli and S. Martello (eds.), *Annotated Bibliographies in Combinatorial Optimization*, John Wiley and Sons (1997) 393–413.

[25] T. Fahle, U. Junker, S.E. Karisch, N. Kohl, M. Sellmann and B. Vaaben, "Constraints programming based column generation for crew assignment," *Journal of Heuristics* 8 (2002) 59–81.

[26] A.A. Farley and K.V. Richardson, "Fixed charge problems with identical fixed charges," *European Journal of Operational Research* 18 (1984) 245–249.

[27] T.A. Feo, M.G.C. Resende and S.H. Smith, "A greedy randomized adaptive search procedure for maximum independent set," *Operations Research* 42 (1994) 635–643.

[28] M.L. Fisher, "The Lagrangian relaxation method for solving integer programming problems," *Management Science* 43 (1997) 1520–1536.

[29] H. Foerster and G. Wäscher, "Pattern reduction in one-dimensional cutting stock problems," *International Journal of Production Research* 38 (2000) 1657–1676.

[30] L.M. Gambardella, É.D. Taillard and M. Dorigo, "Ant colonies for the quadratic assignment problem," *Journal of Operations Research Society* 50 (1999) 167–176.

[31] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.H.Freeman and Company (1979).

[32] T. Gau and G. Wäscher, "CUTGEN1: a problem generator for the standard one-dimensional cutting stock problem," *European Journal of Operational Research* 84 572–579.

[33] M. Gendreau, G. Laporte and J.Y. Potvin, "Vehicle routing: modern heuristics," in: *Local Search in Combinatorial Optimization*, E.H.L. Aarts and J.K. Lenstra (eds.), John Wiley and Sons (1997) 311–336.

[34] P.C. Gilmore and R.E. Gomory, "A linear programming approach to the cutting-stock problem," *Operations Research* 9 (1961) 849–859.

[35] P.C. Gilmore and R.E. Gomory, "A linear programming approach to the cutting-stock problem — part II," *Operations Research* 11 (1963) 863–888.

[36] F. Glover, "Tabu search — Part I," *ORSA Journal on Computing* 1 (1989) 190–206; "Part II", ditto 2 (1990) 4–32.

[37] F. Glover, "Genetic algorithms and scatter search: unsuspected potentials," *Statistics and Computing* 4 (1994) 131–140.

[38] F. Glover, "Scatter search and star-paths: beyond the genetic metaphor," *OR Spektrum* 17 (1995) 125–137.

[39] F. Glover, "Ejection chains reference structures and alternating path methods for traveling salesman problems," *Discrete Applied Mathematics* 65 (1996) 223–253.

[40] F. Glover and G.A. Kochenberger, "Critical events tabu search for multidimensional knapsack problems," in: I.H. Osaman and J.P. Kelly (eds.), *Meta-Heuristics: The Theory and Applications*, Kluwer Academic Publisher (1996) 407–427.

[41] F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers (1997).

[42] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley (1989).

[43] C. Goulimis, "Optimal solutions for the cutting stock problem," *European Journal of Operational Research* 44 (1990) 197–208.

[44] M. Gradisar, M. Kljajić, G. Resinovic and J. Jesenko, "A sequential heuristic procedure for one-dimensional cutting," *European Journal of Operational Research* 114 (1999) 557–568.

[45] J. Gu and X. Huang, "Efficient local search with search space smoothing: a case study of the traveling salesman problem (TSP)," *IEEE Transactions on Systems, Man, and Cybernetics* 24 (1994) 728–735.

[46] R.W. Haessler, "A heuristic programming solution to a nonlinear cutting stock problem," *Management Science* 17(12) (1971) 793–802.

[47] R.W. Haessler, "Controlling cutting pattern changes in one-dimensional trim problems," *Operations Research* 23(3) (1975) 483–493.

[48] R.W. Haessler, "A note on computational modifications to the Gilmore-Gomory cutting stock algorithm," *Operations Research* 28(4) (1980) 1001–1005.

[49] R.W. Haessler and P.E. Sweeney, "Cutting stock problems and solution procedures," *European Journal of Operational Research* 54 (1991) 141–150.

[50] S. Hanafi and A. Freville, "An efficient tabu search approach for the 0-1 multidimensional knapsack problem," *European Journal of Operational Research* 106 (1998) 659–675.

[51] P. Hansen and N. Mladenović, "An introduction to variable neighborhood search," in: S. Voß, S. Martello, I.H. Osman and C. Roucairol (eds.), *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers (1999) 433–458.

[52] A.I. Hinxman, "The trim-loss and assortment problems: a survey," *European Journal of Operational Research* 5 (1980) 8–18.

[53] K.L. Hoffman and M. Padberg, "Solving airline crew scheduling problems by branch-and-cut," *Management Science* 39(6) (1993) 657–682.

[54] J.H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, The University of Michigan Press (1975), and MIT Press (1992).

[55] D.S. Johnson, C.R. Aragon, L.A. McGreoch and C. Schevon, "Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning," *Operations Research* 37 (1989) 865–892; "part II, graph coloring and number partitioning," ditto 39 (1991) 378–406.

[56] D.S. Johnson, "Local optimization and the traveling salesman problem," in: M.S. Paterson (eds.), *Automata, Languages and Programming, Lecture Notes in Computer Science* 443 (1990) 446–461.

[57] D.S. Johnson and L.A. McGeoch, "The traveling salesman problem: a case study," in: E.H.L. Aarts and J.K. Lenstra (eds.), *Local Search in Combinatorial Optimization*, John Wiley and Sons (1997).

[58] R.E. Johnston, "Rounding algorithm for cutting stock problems," *Journal of Asian-Pacific Operations Research Societies* 3 (1986) 82–92.

[59] L.V. Kantorovich, "Mathematical methods of organising and planning production," *Management Science* 6 (1960) 366–422.

[60] B.W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal* 49 (1970) 291–307.

[61] G.A.P. Kindervater, "Vehicle routing: handling edge exchanges," in: *Local Search in Combinatorial Optimization*, E.H.L. Aarts and J.K. Lenstra (eds.), John Wiley and Sons (1997) 337–360.

[62] S. Kirkpatrick, C.D. Gelatt, Jr. and M.P. Vecchi, "Optimization by simulated annealing," *Science* 220 (1983) 671–680.

[63] M. Laguna, T.A. Feo and H.C. Elrod, "A greedy randomized adaptive search procedure for the two-partition problem," *Operations Research* 42 (1994) 677–687.

[64] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys (eds.), *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, John Wiley and Sons (1985).

[65] S. Lin and B.W. Kernighan, "An efficient heuristic algorithm for the traveling salesman problem," *Operations Research* 21 (1973) 498–516.

[66] R.A. Murphey, P.M. Pardalos and M.G.C. Resende, "Frequency assignment problems," in: *Handbook of Combinatorial Optimization*, Kluwer Academic Publishers (1999).

[67] C. McDiarmid, "Pattern minimisation in cutting stock problems," *Discrete Applied Mathematics* 98 (1999) 121–130.

[68] O. Marcotte, "The cutting stock problem and integer rounding," *Mathematical Programming* 33 (1985) 82–92.

[69] O.C. Martin, S.W. Otto and E.W. Felten, "Large-step markov chains for the TSP incorporating local search heuristic," *Operations Research Letters* 11 (1992) 219–224.

[70] N. Megiddo, "Linear programming in linear time when the dimension is fixed," *Journal of the ACM* 31 114–127.

[71] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers and Operations Research* 24 (1997) 1097–1100.

[72] G.L. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley and Sons (1988).

[73] I.H. Osman and J.P. Kelly (eds.), *Meta-Heuristics: Theory and Applications*, Kluwer Academic Publishers (1996).

[74] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and complexity*, Printice-Hall (1982).

[75] E. Pesch and F. Glover, "TSP ejection chains," *Discrete Applied Mathematics* 76 (1997) 165–181.

[76] J.F. Pierce, "On the solution of integer cutting stock problems by combinatorial programming, Part I," *IBM Technical Report* 36.Y02, Cambridge Scientific Center, Cambridge, MA (1966).

[77] C.R. Reeves, *Modern Heuristics Techniques for Combinatorial Problems*, Black-well Scientific Publications (1993); re-issued by McGraw-Hill (1995).

[78] S. Sahni, "Approximates for the 0/1 knapsack problem," *Journal of the Association for Computing Machinery* 22 (1) (1975) 115–124.

[79] G. Scheithauer and J. Terno, "The modified integer round-up property of the one-dimensional cutting stock problem," *European Journal of Operational Research* 84 (1995) 562–571.

[80] G. Scheithauer and J. Terno, "A branch-and-bound algorithm for solving one-dimensional cutting stock problems exactly," *Applied Mathematics* 23 (1995) 151–167.

[81] G. Scheithauer and J. Terno, "Theoretical investigations on the modified integer round-up property for the one-dimensional cutting stock problem," *Operations Research Letters* 20 (1997) 93–100.

[82] D.H. Smith, S. Hurley and S.U. Thiel, "Improving heuristics for the frequency assignment problem," *European Journal of Operational Research* 107 (1998) 76–86.

[83] H. Stadtler, "A one-dimensional cutting stock problem in the aluminum industry and its solution," *European Journal of Operational Research* 44 (1990) 209–223.

[84] E. Sweeney and R. W. Haessler, "One-dimensional cutting stock decisions for rolls with multiple quality grades," *European Journal of Operational Research* 44 (1990) 224–231.

[85] T. Terlaky, "A convergent Criss-cross method," *Mathematische Operationsforscheung und Statistics ser. Optimization* 16 (1985) 683–690.

[86] E. Tsang and C. Voudouris, "Fast local search and guided local search and their application to British Telecom's workforce scheduling problem," *Operations Research Letters* 20 (1997) 119–127.

[87] S. Umetani, M. Yagiura and T. Ibaraki, "One dimensional cutting stock problem to minimize the number of different cutting patterns," *European Journal of Operational Research* 146(2) (2003) 388–402.

[88] S. Umetani, M. Yagiura and T. Ibaraki, "A local search approach to the pattern restricted one dimensional cutting stock problem," submitted for publication.

[89] S. Umetani, M. Yagiura and T. Ibaraki, "An LP-based local search to the one dimensional cutting stock problem using a given number of cutting patterns," to appear *The IEICE Transactions on Fundamentals.*

[90] S. Umetani, M. Yagiura and T. Ibaraki, "A local search approach for one dimensional cutting stock problem," *Proceedings of 4th Metaheuristics International Conference* (MIC2001) (2001) 69–73.

[91] S. Umetani, M. Yagiura and T. Ibaraki, "An LP-based local search to the one dimensional cutting stock problem using a given number of cutting patterns," *Proceedings of the 1st International Workshop on Heuristics* (IWH2002) (2002) 28–38.

[92] R. Vahrenkamp, "Random search in the one-dimensional cutting stock problem," *European Journal of Operational Research* 95 (1996) 191–200.

[93] P.H. Vance, "Branch-and-price algorithms for the one-dimensional cutting stock problem," *Computational Optimization and Applications* 9 (1998) 211–228.

[94] F. Vanderbeck, "Computational study of a column generation algorithm for bin packing and cutting stock problems," *Mathematical Programming* 86 (1999) 565–594.

[95] F. Vanderbeck, "Exact algorithm for minimising the number of setups in the one-dimensional cutting stock problem," *Operations Research* 48 (2000) 915–926.

[96] C. Voudouris and E. Tsang, "Guided local search and its application to the traveling salesman problem," *European Journal of Operational Research* 113 (1999) 469–499.

[97] W.E. Walker, "A heuristic adjacent extreme point algorithm for the fixed charge problem," *Management Science* 22(5) (1976) 587–696.

[98] G. Wäscher and T. Gau, "Heuristics for the integer one-dimensional cutting stock problem: a computational study," *OR Spektrum* 18 (1996) 131–144.

[99] L.A. Wolsey, *Integer Programming*, John Wiley and Sons (1998).

[100] M. Yagiura, T. Ibaraki and F. Glover, "An ejection chain approach for the generalized assignment problem," Technical Report #99013, Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University (1999).

[101] M. Yagiura and T. Ibaraki, "On metaheuristic algorithms for combinatorial optimization Problems," *Systems and Computers in Japan* 32(3) (2001) 33–55.

[102] M. Yagiura and T. Ibaraki, "Local search," in: P.M. Pardalos and M.G.C. Resende (eds.), *Handbook of Applied Optimization*, Oxford University Press (2002) 104–123.

[103] S. Zionts, "The criss-cross method for solving linear programming problems," *Management Science* 15 (1969) 426–445.